ISTQB® Certified Tester
Foundation Level
Alapszintű tanúsítvány
Hivatalos magyar nyelvű tanterv

Verzió: 4.0.1, 2024.03.01.

(Official ISTQB® CTFL Syllabus – Hungarian)

(ISTQB® CTFL Syllabus version: V4.0)

Magyar Szoftvertesztelői Tanács Egyesület

Balatoni út 2/a. B. épület 1. emelet

H-1112 Budapest, Hungary

Tel: +36 30 611-4242

info@hstqb.org

Dokumentum verziókövetés

Verzió	Dátum	Megjegyzés
HTB-2009-1.0	2009-10-06	Első hivatalosan publikált magyar változat az alábbi eredeti változat alapján. Teendők: függelékek megírása.
ISTQB® 2007	2009-05-01	Certified Tester Foundation Level Syllabus Maintenance Release
ISTQB® 2007 1.1	2009-11-15	Kisebb javítások, a Syllabus 2.0 változások átvezetése
ISTQB® 2007 1.2	2009-11-22	Kisebb javítások, a Syllabus 2.0 változások átvezetése
ISTQB® 2007 1.3	2010-02-26	A HTB felülvizsgálatának eredménye. A glosszárium 2.1 verziójával való összehangolás
ISTQB® 2007 2.0	2010-03-15	A HTB felülvizsgálatának eredménye. Szinkronizálás a Szoftvertesztelés egységesített kifejezéseinek gyűjteménye (glossary) 3.0 verzióval, amellyel együtt kerül kiadásra.
ISTQB® 2010 2.1	2011-02-14	A Foundation+Level+Syllabus+(2010).pdf angol nyelvű tanterv változásainak, illetve a kifejezésgyűjtemény (glosszárium) változásainak átvezetése
ISTQB® 2011 3.0	2011-05-04	A HTB által felülvizsgált változat. Szinkronban a Foundation+Level+Syllabus+(2011).pdf angol nyelvű tantervvel, illetve a Szoftvertesztelés egységesített kifejezéseinek gyűjteménye 3.1 verzióval
ISTQB® 2011, HTB 3.01	2011-11-13	Kisebb javítások
ISTQB® 2011, HTB 3.02	2012-02-28	rendszer teszt –> rendszerteszt halott kód –> elérhetetlen kód hibaarány –> meghibásodási ráta hibák csoportosulása -> hibafürtök megjelenése teljesítmény teszt -> teljesítményteszt hibajelenlét kimutatása -> hibák látszólagos hiánya a hibátlan rendszer téveszméje -> a hibamentes rendszer téveszméje tesztelési alapelv (test policy) –> tesztelési irányelvek a „tesztelés” szó több helyen „teszt”-re változtatva, pl: tesztelés becslése –> tesztbecslés tesztelési stratégia –> tesztstratégia tesztelési feltétel -> tesztfeltétel tesztelési folyamat -> tesztfolyamat tesztcél-> tesztcél tesztelési technika -> teszttechnika tesztelési szint -> tesztszint tesztelés tárgyköre (test object) -> teszt tárgya
ISTQB® 2011, HTB 3.03	2017-08-02	3.2.1 fejezet pontosítása (tördelés)
ISTQB® 2018, HTB 1.00	2019-04-19	Az angol nyelvű, Certified Tester Foundation Level Syllabus, Version 2018 Syllabus fordítása
ISTQB® 2018 V3.1, HTB 1.10	2019-12-31	Az ISTQB® által kiadott 2018 V3.1-es frissítés feldolgozása Az eddig beérkezett visszajelzések alapján kisebb javítások
V4.0	2023-11-01	Az ISTQB® által kiadott angol nyelvű Certified Tester Foundation Level V4.0 Syllabus fordítása.

V4.0.1	2024-03-01	Hibajavítások a magyar nyelvű kiadásban.

Tartalomjegyzék

Köszönetnyilvánítás

A magyar változat elkészítésében közreműködtek: Albert Laura, Gergely Tamás, Horváth Andrea, Horváth Ágota, Horváth Tamás, Mátyás Márton, Medzihradszky Dénes, Orosz Dénes, Siska Márton.

A korábbi magyar változatok elkészítésében közreműködtek: Albert Laura, Csapó Ákos György, Gillespie Raymond, Frei Dávid, Horváth Ágota, Horváth Tamás, Mátyás Márton, Medzihradszky Dénes, Csonka Béla, Beszédes Árpád, Kapros Gábor, Kovács Attila, Kusper Gábor, Stöckert Tamás.

0. Bevezetés

0.1 A hivatalos magyar nyelvű tanterv célja

Jelen tanterv alapot nyújt az International Software Testing Qualifications Board (ISTQB®) által hivatalosan jóváhagyott Alapszintű Tesztelői Tanúsítvány (Certified Tester Foundation Level (CTFL)) minősítéshez.

Jelen tanterv a Magyar Szoftvertesztelői Tanács Egyesület (Hungarian Testing Board – HTB) által hivatalosan kiadott, az eredeti angol nyelvű tanterv magyar nyelvű fordítása, a hivatalosan elfogadott magyar kifejezésgyűjtemény (ISTQB® Glosszárium) alapján.

A tantervet a HTB a nemzeti vizsgáztató szerv (HTB) részére vizsgakérdések saját nyelven történő kidolgozásához, valamint a képzési szolgáltatók akkreditációjához bocsájtja rendelkezésre. A képzési szolgáltatók biztosítják a tanfolyam anyagát, és meghatározzák a megfelelő oktatási módszereket az akkreditációhoz; a tanterv pedig segíti a tanulókat a vizsgára való felkészülésben.

0.2 Az Alapszintű Tesztelő Tanúsítvány a szoftvertesztelésben

Az Alapszintű Tanúsítványt bárki megszerezheti, aki szoftvertesztéléssel foglalkozik. Ebbe a körbe tartoznak a tesztelők, tesztelemzők, tesztmérnökök, tesztelési tanácsadók, tesztmenedzserek, szoftverfejlesztők és fejlesztői csapattagok. Az Alapszintű Tanúsítvány azok számára is kívánatos lehet, akik a szoftvertesztelés alapjait szeretnék megismerni: terméktulajdonosok, projektmenedzserek, minőségirányítók, szoftverfejlesztő menedzserek, üzleti elemzők, IT vezetők és menedzsment tanácsadók. Az Alapszintű Tanúsítvánnyal rendelkezők számára lehetőség nyílik magasabb szintű szoftvertesztelési tanúsítvány megszerzésére.

0.3 Karrierút a tesztelők számára

Az ISTQB® tanúsítási rendszer támogatást nyújt a szoftvertesztelő szakemberek számára karrierjük minden szakaszában, széleskörű és mélyreható tudást nyújtva. Azokat, akik megszerezték a CTFL tanúsítványt, talán további tanúsítványok is érdekelhetik, úgy, mint a Core Advanced szintű Test Analyst (ISTQB-CTAL-TA), Technical Test Analyst (ISTQB-CTAL-TTA) és a Test Management (ISTQB-CTAL-TM), majd az Expert szintű tanúsítványok (Improving the Test Process (ISTQB-CTEL-ITP) vagy a Test Management (ISTQB-CTEL-TM)) tanúsitvány. Bárki, aki Agilis környezetben alkalmazott tesztelési gyakorlatok fejlesztésében érdekelt, fontolóra veheti az Agile Technical Tester (ISTQB-ATT) vagy az Agile Test Leadership at Scale (ISTQB-ATLAS) tanúsítványokat. A specialista (Specialist) irány olyan területekben mélyed el, amelyek specifikus tesztelési megközelítésekkel és tesztelési tevékenységekkel foglalkoznak (pl. Teszt automatizáció (ISTQB-CT-TAE), AI tesztelés (ISTQB-CT-AI), Modell Alapú Tesztelés (ISTQB-CT-MBT) ill. Mobilalkalmazás-tesztelés (ISTQB-CT-MAT)), vagy amelyek specifikus tesztelési területekkel foglalkoznak (pl. Teljesítménytesztelés (ISTQB-CT-PT), Használhatósági Tesztelés (ISTQB-CT-UT), Elfogadási Tesztelés (ISTQB-CT-ACT), Biztonsági Tesztelés (ISTQB-CT-SEC)), vagy amelyek bizonyos iparági szegmensekhez szorosan kapcsolódó tesztelési ismereteket gyűjtenek össze (pl. Autóipar (ISTQB-CT-AUT) vagy Játékipar (ISTQB-CT-GAME)). A legfrissebb információkért az ISTQB Certified Tester rendszerről látogasson el a www.istqb.org weboldalra.

0.4 Üzleti eredmények

Ez a fejezet tartalmazza azt a 14 üzleti eredményt, amely elvárható egy alapszintű minősítéssel rendelkező személytől:

Az alapszintű minősített tesztelő…

FL-BO1 Megérti, mi az a tesztelés és miért előnyös az.

FL-BO2 Megérti a szoftvertesztelés alapvető fogalmait.

FL-BO3 A tesztelési megközelítést és tevékenységeket a tesztelési kontextustól függően azonosítja és hajtja végre.

FL-BO4 Értékeli és javítja a dokumentáció minőségét.

FL-BO5 Növeli a tesztelés hatásosságát és hatékonyságát.

FL-BO6 A tesztelési folyamatot a szoftverfejlesztési életciklushoz igazítja.

FL-BO7 Megérti a tesztmenedzsment alapelveit.

FL-BO8 Világos és érthető hibajelentéseket ír és fogalmaz meg.

FL-BO9 Megérti a teszteléssel kapcsolatos prioritásokat és ráfordításokat befolyásoló tényezőket.

FL-BO10 Egy többfunkcionális csapat részeként dolgozzon.

FL-BO11 Ismeri a tesztautomatizálás kockázatait és előnyeit.

FL-BO12 Azonosítja a teszteléshez szükséges alapvető készségeket.

FL-BO13 Megérti a kockázat tesztelésre gyakorolt hatását.

FL-BO14 Hatékonyan jelent a tesztelés haladásáról és minőségéről.

0.5 A vizsga tárgyát képező tanulási célok és kognitív tudásszintek

A tanulási célok támogatják az üzleti eredményeket, valamint ezek alapján készülnek az Alapszintű Tanúsítvány vizsga kérdés sorai.

Általánosságban elmondható, hogy jelen tanterv minden fejezete a vizsga tárgyát képezheti a K1 tudásszinten, kivéve a bevezetőt, illetve a mellékleteket. Ez azt jelenti, hogy a tanterv által, a hat fejezet bármelyikében említett kulcsszavakat vagy koncepciókat a jelöltnek fel kell ismernie, fel kell tudni idéznie és emlékeznie kell rájuk. A tanulási célokhoz rendelt tudásszintek minden fejezet elején felsorolásra kerülnek és az alábbiak szerint vannak osztályozva:

- K1: emlékezés
- K2: megértés
- K3: alkalmazás

A tanulási célokkal összefüggő további példák és tanulási célok az A függelékben találhatók.

A fejezetek címei alatt lévő „Kulcsszavak” bekezdésekben felsorolt szakkifejezésekre emlékezni kell (K1) akkor is, ha a tanulási célok ezt külön nem említik meg.

0.6 Az Alapszintű Tanúsítvány vizsga

Az Alapszintű Tanúsítványhoz kapcsolódó vizsga az itt közölt tantervre épül. A vizsgakérdések megválaszolásához a tanterv több fejezetének ismeretére is szükség lehet. A tanterv bármely fejezete vizsga tárgyát képezheti, kivéve a bevezetőt, illetve a függelékeket. Szabványok és szakkönyvek hivatkozásként szerepelnek (7. fejezet), de azok tartalma nem képezheti a vizsga tárgyát azon túlmenően, amit a jelen tanterv összefoglal az érintett szabványokat, illetve szakkönyveket illetően.

A vizsga feleletválasztós feladatokat tartalmaz. Egy vizsga 40 kérdésből áll. A sikeres vizsgához legalább a kérdések 65%-át (26 kérdés) helyesen kell megválaszolni.

A vizsga letehető akkreditált képzés részeként vagy egyénileg. Magyarországon az Alapszintű Tanúsítvány megszerzéséhez szükséges vizsgát a HTB vizsgaközpontjainál lehet letenni. Akkreditált tanfolyam elvégzése nem előfeltétele a vizsga letételének.

0.7 Akkreditáció

Egy, az ISTQB® által elismert nemzeti bizottság jogosult akkreditálni azokat a képzési szolgáltatókat, amelyek tanterve követi a jelen tantervet. Az akkreditáció irányelveit az akkreditációt végrehajtó bizottságtól vagy testülettől lehet beszerezni. Az akkreditált tanfolyamokat elismerik, mint a jelen tantervnek megfelelőt, és így lehetőség van a tanfolyam részeként vizsgáztatni. A jelen tanterv akkreditációs irányelvei a Folyamatkezelési és Megfelelőségi Munkacsoport (Processes Management and Compliance Working Group) által közzétett általános akkreditációs irányelveket követik.

0.8 Szabványok kezelése

Ez a tanterv több szabványra is hivatkozik (pl. IEEE vagy ISO szabványokra). Ezek a hivatkozások keretrendszert nyújtanak (például az ISO 25010-re vonatkozó hivatkozások a minőségi jellemzők vonatkozásában), vagy további információforrást nyújthatnak, ha az olvasó ezt kívánja. A szabványdokumentumok további tartalma nem részei a vizsgának. A szabványokról további információért lásd a 7. fejezetet.

0.9 Naprakészség

A szoftverfejlesztési ipar gyorsan változik. A változások kezelése és a releváns és aktuális információkhoz való hozzáférés biztosítása érdekében az ISTQB munkacsoportok a www.istqb.org weboldalon hivatkozásokat hoztak létre, amelyek a támogató dokumentációra és a szabványok változásaira utalnak. Ezek az információk nem részei a vizsgának.

0.10 Részletesség

A tanterv részletessége nemzetközi szinten egységes oktatást és vizsgáztatást tesz lehetővé. Ennek érdekében a tanterv a következőkből áll:

- Általános képzési célok, melyek az alapszint célkitűzéseit tartalmazzák.
- Kifejezések listája, melyeket a hallgatóknak képesnek kell lenniük felidézni.
- Az egyes területek oktatási céljai, melyek az elérendő kognitív tanulási eredményt és tudást írják le.
- Az oktatandó alapfogalmak leírása, beleértve a hivatkozásokat az elismert forrásokhoz

A tanterv nem nyújt teljes leírást a szoftvertesztelésről; csak az alapszintű képzések részletességével érinti a témát. Azon tesztfogalmakra és technikákra koncentrál, melyek minden projektre alkalmazhatók, függetlenül az alkalmazott szoftverfejlesztési életciklustól.

0.11 A tanterv felépítése

A tanterv hat, a vizsga tárgyát is képező fejezetet tartalmaz. A címsor tartalmazza az adott fejezetre szánt időt. A fejezetszint alatt nincsenek meghatározott időkorlátok. Az akkreditált tanfolyamok számára jelen tanterv minimum 1135 perces (18 óra és 55 perc) oktatási tervet vár el a hat fejezetre az alábbiak szerint elosztva:

- 1. Fejezet: A tesztelés alapjai (180 perc)
  - A teszteléssel kapcsolatos alapelvek, a tesztelés szükségességének okai és a tesztcélok megismerése.
  - A tesztfolyamat, a főbb teszttevékenységek és a tesztver megértése.
  - A teszteléshez szükséges alapvető készségek megismerése és megértése.
- 2. Fejezet: Tesztelés a szoftverfejlesztés életciklusán át (130 perc)
  - A tesztelés különböző fejlesztési megközelítésekbe történő integrálásának megismerése.
  - A "test-first" megközelítések és a DevOps koncepciójának megismerése.
  - A különböző tesztszintek, teszttípusok és a karbantartási tesztelés megismerése.
- 3. Fejezet: Statikus tesztelés (80 perc)
  - A statikus tesztelés alapjainak, a visszajelzési, valamint felülvizsgálati folyamatok megismerése.
- 4. Fejezet: Tesztelemzés és műszaki tervezés (390 perc)
  - A feketedoboz-, fehérdoboz- és tapasztalatalapú teszttechnikák alkalmazása a különböző szoftveres munkatermékekből származó tesztesetek levezetésére.
  - Az együttműködés-alapú tesztelési megközelítés megismerése.
- 5. Fejezet: Teszttevékenységek menedzselése (335 perc)
  - A tesztek tervezésének és a teszt ráfordítási igény becslésének megismerése általánosságban.
  - A kockázatok hatásának megértése a tesztelés hatókörére.
  - A teszttevékenységek felügyeletének és irányításának megismerése.
  - A konfigurációmenedzsment szerepének megértése a tesztelés támogatásában.
  - A világos és érthető hibajelentések megismerése.
- 6. Fejezet: Teszteszközök (20 perc)
  - Eszközök osztályozásának megismerése és a tesztautomatizálás kockázatainak és előnyeinek megértése.

1. A tesztelés alapjai 180 perc

Kulcsszavak

emberi eredetű hiba, hiba, hibakeresés, kiváltó ok, lefedettség, meghibásodás, minőség, minőségbiztosítás, műszaki teszttervezés, tesztadat, tesztbázis, tesztcél, tesztelemzés, tesztelés, teszteljárás, teszteredmény, teszteset, tesztfeltétel, tesztfelügyelet, tesztirányítás, tesztlezárás, tesztmegvalósítás, teszt tárgya, teszttervezés, tesztvégrehajtás, tesztver, validáció, verifikáció

Tanulási célok az első fejezethez:

1.1 Mi a tesztelés?

AK-1.1.1 (K1) Azonosítson tipikus tesztcélokat

AK-1.1.2 (K2) Különböztesse meg a tesztelést és a hibakeresést

1.2 Miért szükséges a tesztelés?

AK-1.2.1 (K2) Szemléltesse a tesztelés szükségességét

AK-1.2.2 (K1) Idézze fel a tesztelés és a minőségbiztosítás közötti kapcsolatot

AK-1.2.3 (K2) Tegyen különbséget a kiváltó ok, az emberi eredetű hiba, hiba és meghibásodás között

1.3 Tesztelési alapelvek

AK-1.3.1 (K2) Ismertesse az általános tesztelési alapelveket (A hét tesztelési alapelv)

1.4 Teszttevékenységek, tesztver és teszt szerepkörök

AK-1.4.1 (K2) Foglalja össze a különböző teszttevékenységeket és feladatokat

AK-1.4.2 (K2) Ismertesse a kontextus tesztfolyamatra gyakorolt hatását

AK-1.4.3 (K2) Tegyen különbséget a teszttevékenységeket támogató tesztverek között

AK-1.4.4 (K2) Ismertesse a nyomonkövethetőség fenntartásának fontosságát

AK-1.4.5 (K2) Hasonlítsa össze a tesztelés különböző szerepköreit

1.5 Alapvető készségek és bevált gyakorlatok a tesztelésben

AK-1.5.1 (K2) Adjon példákat a teszteléshez szükséges készségekre

AK-1.5.2 (K1) Idézze fel a teljes csapat megközelítés előnyeit

AK-1.5.3 (K2) Tegyen különbséget a tesztelés függetlenségének előnyei és hátrányai között

1.1 Mi a tesztelés?

A szoftverrendszerek az élet szerves részét képezik. A legtöbb embernek volt már tapasztalata olyan szoftverrel, ami nem az elvártaknak megfelelően működött. A helytelenül működő szoftver sok problémához vezethet, beleértve a pénz- és időveszteséget, az üzleti hírnév elvesztését, illetve extrém esetekben akár sérülést vagy halált is eredményezhet. A szoftvertesztelés a szoftver minőségét értékeli, illetve segít a szoftver működés közbeni meghibásodási kockázatának csökkentésében.

A szoftvertesztelés olyan tevékenységek összessége, amelyek célja a hibák feltárása és a szoftvertermékek minőségének értékelése. Az ilyen termékeket a tesztelés folyamán a teszt tárgyának nevezzük. Gyakori tévhit a teszteléssel kapcsolatban, hogy az csak tesztek végrehajtásából áll (azaz a szoftver futtatásából és a teszteredmények ellenőrzéséből). A szoftvertesztelés azonban más tevékenységeket is magában foglal, valamint hozzá kell igazítani az adott szoftverfejlesztési életciklushoz (lásd a 2. fejezetet).

Egy másik gyakori félreértés a teszteléssel kapcsolatban, hogy teljes mértékben a teszt tárgyának verifikációjára fókuszál. A tesztelés valóban kiterjed a verifikációra, azaz a meghatározott követelményeknek való megfelelés ellenőrzésére, azonban tartalmazza a validációt is, amely azt ellenőrzi, hogy a rendszer megfelel-e a felhasználói, illetve más érdekelt felek igényeinek a működési környezetben.

A tesztelés lehet dinamikus vagy statikus. A dinamikus tesztelés magában foglalja a szoftver végrehajtását, míg a statikus tesztelés nem. A statikus tesztelés a felülvizsgálatot (lásd a 3. fejezetet) és a statikus elemzést jelenti. A dinamikus tesztelés különböző típusú teszttechnikákat és tesztmegközelítéseket használ a tesztesetek tervezésére (lásd a 4. fejezetet).

A tesztelés nem csak technikai tevékenység. Alaposan meg kell tervezni, menedzselni, becsülni, nyomon követni és ellenőrizni (lásd az 5. fejezetet).

A tesztelők eszközöket használnak (lásd a 6. fejezetet), de fontos megjegyezni, hogy a tesztelés nagyrészt intellektuális tevékenység, amely speciális ismereteket, elemzőkészségeket, valamint kritikus és rendszer-szemléletű gondolkodást kíván meg a tesztelőktől (Myers 2011, Roman 2018).

Az ISO/IEC/IEEE 29119-1 szabvány további információkat nyújt a szoftvertesztelési koncepciókról.

1.1.1 Tesztcélok

A tesztelés jellemző céljai az alábbiak:

- Követelmények, felhasználói történetek, műszaki tervek és a kód kiértékelése
- Meghibásodások okozása és hibák megtalálása
- A szükséges lefedettség biztosítása a teszt tárgyának
- A nem megfelelő szoftverminőség kockázati szintjének csökkentése
- Annak igazolása, hogy bizonyos követelmények teljesültek-e
- Annak igazolása, hogy a teszt tárgya megfelel a szerződésben foglalt, jogi vagy szabályozási követelményeknek
- Információ biztosítása az érdekelt feleknek, hogy ezáltal megalapozott döntéseket hozhassanak
- Bizalom kiépítése a teszt tárgyának minőségével kapcsolatban
- Annak validálása, hogy a teszt tárgya elkészült és az érdekelt felek elvárásainak megfelelően működik.

A tesztelés céljai a környezettől függően változhatnak, mely magába foglalja a tesztelés alatt álló munkaterméket, a tesztszintet, a kockázatokat, a követett szoftverfejlesztési életciklusmodellt és az üzleti környezethez kapcsolódó tényezőket, mint például a vállalati struktúra, a versenyszempontok vagy a piacra jutás ideje.

1.1.2 Tesztelés és hibakeresés

A tesztelés és a hibakeresés különálló tevékenységek. A tesztelés olyan meghibásodásokat indukálhat, amelyeket a szoftver hibái okoznak (dinamikus tesztelés), vagy közvetlenül találhat hibákat a teszt tárgyában (statikus tesztelés).

Amikor a dinamikus tesztelés (lásd a 4. fejezetet) meghibásodást vált ki, a hibakeresés ennek a meghibásodásnak az okait (hibák) tárja fel, elemzi és megszünteti ezen okokat. A tipikus hibakeresési folyamat ebben az esetben a következőket tartalmazza:

- A meghibásodás reprodukálása
- Diagnózis (a kiváltó ok megtalálása)
- Az ok kijavítása

Ezt követően az ellenőrző tesztelés során ellenőrzik, hogy a javítások orvosolták-e a problémát. Lehetőség szerint az ellenőrző tesztelést ugyanaz a személy végzi el, aki az első tesztet is elvégezte. Később regressziós tesztelés is elvégezhető annak ellenőrzésére, hogy a javítások nem okoztak-e hibákat a teszt tárgyának más részein (az ellenőrző és a regressziós teszteléssel kapcsolatos további információkért lásd a 2.2.3. fejezetet).

Amikor a statikus tesztelés hibát azonosít, a hibakeresés közvetlenül annak eltávolítását célozza. Nincs szükség reprodukálásra vagy diagnózisra, mivel a statikus tesztelés közvetlenül találja meg a hibákat, és nem okozhat meghibásodást (lásd a 3. fejezetet).

1.2 Miért szükséges a tesztelés?

A tesztelés, mint a minőségellenőrzés egyik formája, segít a kitűzött célok elérésében a meghatározott hatókör-, idő-, minőség- és költségvetési korlátok között. A tesztelés hozzájárulása a sikerhez nem korlátozódhat a tesztcsoport tevékenységeire. Bármely érdekelt fél felhasználhatja tesztelési készségeit, hogy közelebb vigye a projektet a sikerhez. A komponensek, rendszerek és a kapcsolódó dokumentáció tesztelése segít a szoftverhibák azonosításában.

1.2.1 A tesztelés hozzájárulása a sikerhez

A tesztelés költséghatékony módszert kínál a hibák észlelésére. Ezek a hibák ezután eltávolíthatók (hibakereséssel – ami nem tesztelési tevékenység), így a tesztelés közvetetten hozzájárul a teszt tárgyainak jobb minőségéhez.

A tesztelés lehetőséget biztosít, hogy közvetlenül értékeljük a teszt tárgyának minőségét a szoftverfejlesztési életciklus különböző szakaszaiban. Ezek az intézkedések egy nagyobb projektmenedzsment tevékenység részét alkotják, hozzájárulva a szoftverfejlesztési életciklus következő szakaszába lépésről hozott döntésekhez, mint például a kiadási döntés.

A tesztelés segítségével a felhasználók közvetett módon vehetnek részt a fejlesztési projektben. A tesztelők biztosítják, hogy a felhasználók igényeit a fejlesztési életciklus során figyelembe vegyék. A másik lehetőség, hogy a felhasználók egy reprezentatív körét bevonják a fejlesztési projektbe, viszont ez a magas költségek és a megfelelő felhasználók elérhetőségének hiánya miatt általában nem megoldható.

A tesztelést a szerződéses vagy jogi követelményeknek, illetve a szabályozói szabványoknak való megfelelés érdekében is megkövetelhetik.

1.2.2 Tesztelés és minőségbiztosítás (QA)

Bár az emberek a „tesztelés” és a „minőségbiztosítás” (vagy QA) kifejezéseket szinonimaként használják, a két fogalom nem ugyanaz. A tesztelés a minőségellenőrzés (vagy QC) egyik formája.

A minőségellenőrzés egy termék-orientált, javító jellegű megközelítés, amely a megfelelő minőségi szint elérését támogató tevékenységekre összpontosít. A tesztelés a minőségellenőrzés egyik fontos része, további részei a formális módszerek (modellellenőrzés és a helyesség igazolása), a szimuláció és a prototípus készítés.

A minőségbiztosítás folyamatorientált, megelőző jellegű megközelítés, amely a folyamatok megvalósítására és fejlesztésére összpontosít. Az alapelve, hogy ha egy jó folyamatot helyesen követnek, akkor jó terméket hoz létre. A minőségbiztosítás mind a fejlesztési, mind a tesztelési folyamatra értelmezhető, és a projektben mindenki felelőssége.

A teszteredményeket a minőségbiztosítás és a minőségellenőrzés is használja. A minőségellenőrzésben a hibák javítására szolgálnak, míg a minőségbiztosításban visszajelzést adnak a fejlesztési és tesztelési folyamatok teljesítményéről.

1.2.3 Emberi eredetű hibák, hibák, meghibásodások és kiváltó okok

Emberi hibákat (angolul error, mistake) emberi lények követhetnek el. Ezek hibákat (angolul defect, fault, bug) eredményeznek, amelyek meghibásodásokhoz (angolul failure) vezethetnek. Az emberi eredetű hibáknak különféle okai lehetnek, például az idő szorossága, a munkatermékek, folyamatok, infrastruktúra vagy interakciók összetettsége, vagy egyszerűen a fáradtság, esetleg a nem megfelelő képzettség.

Hibák a dokumentációban találhatók, például a követelményspecifikációban vagy a tesztszkriptben, a forráskódban vagy egy kiegészítő termékben, például egy build fájlban. A szoftverfejlesztési életciklus korai szakaszaiban készült termék hibái, ha nem vesszük észre őket, gyakran hibás termékekhez vezetnek az életciklus későbbi szakaszában. Ha egy kódhiba végrehajtásra kerül, előfordulhat, hogy a rendszer nem tudja megtenni azt, amit tennie kell, vagy megtesz valamit, amit nem kellene, és ezek meghibásodást okoznak. Egyes hibák végrehajtása mindig meghibásodást okoz, míg mások csak adott körülmények között, megint mások pedig soha nem vezetnek meghibásodáshoz.

A meghibásodásoknak nem a hibák és az emberi eredetű hibák a kizárólagos okai. Környezeti feltételek is okozhatják, például amikor a sugárzás vagy az elektromágneses mező hibákat okoz az alapszoftverben.

A kiváltó ok egy probléma (pl. egy hibához vezető helyzet) előfordulásának az alapvető oka. A kiváltó okok azonosítása a kiváltó okok elemzésével történik, amelyet általában meghibásodás vagy hiba azonosításakor hajtanak végre. Általánosan elfogadott, hogy a további hasonló meghibásodások vagy hibák megelőzhetők vagy gyakoriságuk csökkenthető a kiváltó ok kezelésével, például annak eltávolításával.

1.3 Tesztelési alapelvek

Számos tesztelési alapelvet javasoltak az elmúlt években, amelyek általános, mindennemű tesztelésre vonatkozó irányelveket adnak. A tanterv hét ilyen alapelvet ismertet.

1. **A tesztelés a hibák jelenlétét mutatja, nem a hiányukat**

A tesztelés kimutathatja a hibák jelenlétét a teszt tárgyában, de azt nem képes igazolni, hogy nincsenek hibák (Buxton 1970). A teszteléssel csökken annak az esélye, hogy a teszt tárgyában felfedezetlen hibák maradnak, de még ha nem is találnak hibát, az nem bizonyítja a rendszer helyességét.

2. **Nem lehetséges kimerítő teszt**

Mindenre kiterjedő tesztelés a triviális eseteket leszámítva nem lehetséges (Manna 1978). Ahelyett, hogy megkísérelnénk a kimerítő tesztelést, a tesztelési erőforrások összpontosításához alkalmazzunk

teszttechnikákat (lásd a 4. fejezetet), teszteset-priorizálást (lásd az 5.1.5 fejezetet) és kockázatalapú tesztelést (lásd 5.2 szakasz).

3. **A korai tesztelés időt és pénzt spórol**

A folyamat korai szakaszában eltávolított hibák nem okoznak későbbi hibákat a származtatott munkatermékekben. A minőség költsége csökken, mivel a szoftverfejlesztési életciklus későbbi szakaszaiban kevesebb meghibásodás következik be (Boehm 1981). A hibák korai felismerése érdekében mind a statikus tesztelést (lásd a 3. fejezetet), mind a dinamikus tesztelést (lásd a 4. fejezetet) a lehető legkorábban el kell kezdeni.

4. **Hibafürtök megjelenése**

A kiadást megelőző tesztelés során megtalált hibák többsége rendszerint néhány rendszer komponensben koncentrálódik, vagy ezen modulok felelősek a működési meghibásodások többségéért (Enders 1975). Ez a jelenség a Pareto-elvet szemlélteti. A megjósolt hibafürtök és a tesztelés vagy működés során ténylegesen megfigyelt hibafürtök fontos bemenetként szolgálnak a kockázatalapú tesztelés számára (lásd az 5.2 fejezetet).

5. **A tesztek elkopnak**

Ha ugyanazokat a teszteket ismételjük számos alkalommal, akkor ezen tesztek nem lesznek hatékonyak új hibák megtalálásában (Beizer 1990). Ahhoz, hogy elkerüljük ezt a hatást, a létező teszteket és tesztadatokat módosítani kell, illetve új teszteket kell írni. Ugyanakkor, néhány esetben - például az automatikus regressziós tesztelés esetében - előnyös oldala is van ugyanazon tesztek ismétlésének (lásd a 2.2.3 fejezetet).

6. **A tesztelés függ a körülményektől**

A tesztelésnek nincs egyetlen univerzálisan alkalmazható megközelítése. A tesztelés különböző környezetekben eltérő módon történik (Kaner 2011).

7. **A hibamentesség téveszméje**

Tévedés (vagy tévhit) azt várni, hogy a szoftververifikáció biztosítja a rendszer sikerét. Az összes meghatározott követelmény alapos tesztelése és a feltárt hibák kijavítása még mindig produkálhat olyan rendszert, amely nem felel meg a felhasználók igényeinek és elvárásainak, amely nem segíti az ügyfél üzleti céljainak elérését, és a versenytársak rendszereihez képest gyengébb. A verifikáció mellett a validálást is el kell végezni (Boehm 1981).

1.4 Teszttevékenységek, tesztver és tesztszerepkörök

A tesztelés függ a környezettől, de magas szinten léteznek gyakori teszttevékenységek, amik nélkül kevésbé valószínű, hogy a tesztelés eléri a tesztcélokat. Ezen teszttevékenységek alkotják a tesztfolyamatot. A tesztfolyamat különböző tényezők alapján az adott helyzetre szabható. Általában az adott helyzetre vonatkozó teszttervezés részeként dől el, hogy a tesztfolyamat melyik teszttevékenységeket foglalja magában, hogyan implementálják ezeket, és mikor valósulnak meg.

A következő szakaszok a tesztfolyamat általános vonatkozásait írják le a teszttevékenységek és -feladatok, a környezet hatása, a tesztverek, a tesztbázis és a tesztverek közötti nyomonkövethetőség, valamint a tesztelési szerepkörök tekintetében.

Az ISO/IEC/IEEE 29119-2 szabvány további információkat nyújt a tesztfolyamatokról.

1.4.1 Teszttevékenységek és -feladatok

A tesztfolyamat általában a lentebb bemutatott fő tevékenységcsoportokból áll. Bár ezen tevékenységek közül több logikailag szekvenciálisnak tűnhet, gyakran iteratív módon vagy párhuzamosan kerülnek megvalósításra. Ezeket a teszttevékenységeket gyakran szükséges lehet a rendszerhez vagy a projekthez igazítani.

A **teszttervezés** részét képezi a tesztcélok meghatározása, majd egy olyan megközelítés kiválasztása, amely a legjobban eléri ezeket a célokat a környezet által támasztott korlátokon belül. A teszttervezést bővebben az 5.1 fejezetben tárgyaljuk.

Tesztfelügyelet és -irányítás. A tesztfelügyelet magában foglalja minden teszttevékenység folyamatos ellenőrzését, valamint az aktuális és a tervezett előrehaladás összehasonlítását. A tesztirányítás a teszt célok eléréséhez szükséges tevékenységeket foglalja magában. A tesztfelügyelet és -irányítás az 5.3 fejezetben kerül bővebben kifejtésre.

A **tesztelemzés** során a tesztbázist elemezzük annak érdekében, hogy azonosítsuk a tesztelhető funkcionalitásokat, valamint meghatározzuk és priorizáljuk a kapcsolódó tesztfeltételeket a kapcsolódó kockázatokkal és azok kockázati szintjével együtt (lásd az 5.2 fejezetet). Értékeljük a tesztbázist és a teszt tárgyát, hogy azonosítsuk a bennük előforduló hibákat, és megállapítsuk a tesztelhetőségüket. A tesztelemzést gyakran támogatják a teszttechnikák alkalmazásával (lásd a 4. fejezetet). A tesztelemzés választ ad a „mit teszteljünk?” kérdésre, mint mérhető lefedettségi kritérium.

Műszaki teszttervezés során a tesztfeltételekből teszteseteket és egyéb tesztvereket alakítunk ki (például tesztvázlatokat). Ez a tevékenység gyakran magában foglalja a lefedettségi elemek azonosítását, amelyek útmutatóul szolgálnak a tesztesetek bemeneteinek meghatározásához. Teszttechnikákat (lásd a 4. fejezetet) használhatunk a tevékenység támogatására. A teszttervezés szintén magában foglalja a tesztadatokra vonatkozó követelmények meghatározását, a tesztkörnyezet megtervezését és egyéb szükséges infrastruktúrák és eszközök azonosítását is. A teszttervezés választ ad a „hogyan teszteljünk?” kérdésre.

A **tesztmegvalósítás** során létrehozzuk vagy beszerezzük a tesztvégrehajtáshoz szükséges tesztvert (például a tesztadatokat). A tesztesetek teszteljárásokba szervezhetők, és gyakran tesztkészletté állnak össze. Kézi és automatizált tesztszkriptek jönnek létre. A teszteljárások prioritást kapnak, és bekerülnek a tesztvégrehajtási ütemtervbe a hatékony tesztvégrehajtás érdekében (lásd az 5.1.5. fejezetet). Elkészül a tesztkörnyezet, melyet ellenőrzünk, hogy megfelelően legyen beállítva.

A **tesztvégrehajtás** során a teszteket a tesztvégrehajtási ütemtervben meghatározott sorrendben hajtjuk végre (tesztfuttatás). A teszt végrehajtása lehet manuális vagy automatizált. A tesztvégrehajtás számos formát ölthet, beleértve a folyamatos tesztelést vagy a páros tesztelési szakaszokat. A tényleges teszteredményeket összehasonlítjuk a várt eredményekkel. A teszteredmények naplózásra kerülnek. Az anomáliákat elemezzük, ezáltal azonosítjuk valószínű okukat. Ez az elemzés lehetővé teszi számunkra, hogy a megfigyelt meghibásodások alapján jelentsük az anomáliákat (lásd az 5.5. fejezetet).

A **tesztlezárás**hoz kapcsolódó tevékenységeket projektmérföldkövek elérésekor hajtjuk végre (például kiadás, egy tesztszint befejezése), a lezáratlan hibákra változtatáskérést vagy termék-teendőlista elemet vehetünk fel. Minden olyan tesztvert, amely hasznos lehet a jövőben, azonosítunk és archiválunk, vagy átadunk a megfelelő csapatoknak. A tesztkörnyezetet egy előre meghatározott állapotba állítjuk. A teszttevékenységeket elemezzük, hogy meghatározzuk a jövőbeli iterációkhoz, kiadásokhoz és projektekhez szükséges változtatásokat (lásd a 2.1.6. fejezetet). Összefoglaló tesztjelentés készül, amelyet az érdekelt felek is megismernek.

1.4.2 Tesztfolyamat és környezete

A tesztelést nem elszigetelten végzik. A tesztelési tevékenységek a szervezeten belüli fejlesztési folyamatok szerves részét képezik. A tesztelést az érdekeltek finanszírozzák, és végső célja, hogy segítse az érdekeltek üzleti igényeinek kielégítését. Ennek következtében a tesztelés kivitelezése számos környezeti tényezőtől függ, beleértve:

- Érdekelt felek (igények, elvárások, követelmények, együttműködési hajlandóság, stb.)
- Csapattagok (készségek, ismeretek, tapasztalati szint, elérhetőség, képzési igények, stb.)

- Üzleti terület (a teszt tárgyának kritikussága, azonosított kockázatok, piaci igények, vonatkozó jogi szabályozás, stb.)
- Technikai tényezők (szoftver típusa, termék architektúrája, használt technológia, stb.)
- A projekt korlátai (hatókör, idő, költségvetés, erőforrások, stb.)
- Szervezeti tényezők (szervezeti felépítés, meglévő szabályzatok, alkalmazott gyakorlatok, stb.)
- Szoftverfejlesztési életciklus (mérnöki gyakorlatok, fejlesztési módszerek, stb.)
- Eszközök (elérhetőség, használhatóság, megfelelőség, stb.)

Ezek a tényezők hatással lehetnek számos teszteléssel kapcsolatos kérdésre, ideértve a következőket: tesztstratégia, alkalmazott teszttechnikák, tesztautomatizálás foka, lefedettség szükséges szintje, a tesztdokumentáció részletessége, jelentések stb.

1.4.3 Tesztver

A tesztver az 1.4.1 szakaszban bemutatott teszttevékenységek kimeneti munkatermékeként keletkezik. Jelentős eltérések mutatkoznak abban, hogy a különböző szervezetek hogyan állítják elő, alakítják, nevezik el, szervezik és kezelik munkatermékeiket. A megfelelő konfigurációmenedzsment (lásd az 5.4 fejezetet) biztosítja a munkatermékek konzisztenciáját és integritását. A munkatermékek nem teljes listája:

- A **teszttervezési munkatermékek** a következők: teszterv, tesztütemterv, kockázati nyilvántartás, valamint belépési és kilépési feltételek (lásd az 5.1. fejezetet). A kockázati nyilvántartás a kockázatok listája, kiegészítve a kockázat valószínűségével, a kockázat hatásával és a kockázatmérséklléssel kapcsolatos információkkal (lásd az 5.2. fejezetet). A tesztütemterv, a kockázati nyilvántartás, valamint a belépési és kilépési feltételek gyakran a teszterv részét képezik.
- A **tesztfelügyelet és -irányítás munkatermékei** a következők: tesztelőrehaladási jelentések (lásd az 5.3.2 fejezetet), irányítási irányelvek dokumentációja (lásd az 5.3 szakaszt) és kockázati információk (lásd az 5.2 fejezetet).
- A **tesztelemzési munkatermékek** a következők: (priorizált) tesztfeltételek (pl. elfogadási feltételek, lásd a 4.5.2. szakaszt), valamint a tesztbázis (nem azonnal javított) hibáira vonatkozó hibajelentések.
- A **műszaki teszttervezés munkatermékei** a következők: (priorizált) tesztesetek, tesztvázlatok, lefedettségi elemek, tesztadatokra vonatkozó követelmények és tesztkörnyezeti követelmények.
- A **tesztmegvalósítás munkatermékek** a következők: teszteljárások, automatizált tesztszkriptek, tesztkészletek, tesztadatok, tesztvégrehajtási ütemterv és a tesztkörnyezet elemei. Példák a tesztkörnyezet elemeire: csonkok, meghajtók, szimulátorok és szolgáltatásvirtualizációk.
- A **tesztvégrehajtási munkatermékei** a következők: tesztnaplók és hibajelentések (lásd az 5.5. fejezetet).
- A **tesztlezárási munkatermékek** a következők: összefoglaló tesztjelentés (lásd az 5.3.2. fejzetet), a következő projektek vagy iterációk fejlesztéséhez megfogalmazott tevékenységek, dokumentált tanulságok és változtatáskérések (pl. termék-teendőlista elemekként).

1.4.4 A tesztbázis és a tesztver közötti nyomonkövethetőség

Ahhoz, hogy hatásosan valósítsuk meg a tesztfelügyelet és -irányítás tevékenységeit, fontos, hogy létrehozzuk és a tesztfolyamat egészén keresztül fenntartsuk a nyomonkövethetőséget a tesztbázis elemei és az ezekhez az elemekhez kapcsolódó tesztverek (pl. tesztkörülmények, kockázatok, tesztesetek), valamint a teszteredmények és az észlelt hibák között.

A pontos nyomonkövethetőség támogatja a lefedettség értékelését, ezért nagyon hasznos, ha mérhető lefedettségi kritériumokat határoznak meg a tesztbázisban. A lefedettségi kritériumok kulcsfontosságú teljesítménymutatókként működhetnek azon tevékenységek számára, amelyek megmutatják, hogy a tesztcélok milyen mértékben valósultak meg (lásd az 1.1.1. szakaszt). Például:

- A tesztesetek nyomonkövethetősége a követelmények felé igazolhatja, hogy a tesztesetek lefedik a követelményeket.
- A teszteredmények nyomonkövethetősége a kockázatok felé felhasználható a teszt tárgyában továbbra is fennálló kockázat szintjének értékelésére.

A jó nyomonkövethetőség a lefedettség értékelése mellett lehetővé teszi a változások hatásának meghatározását, megkönnyíti a tesztauditokat, és segíti az IT-irányítási kritériumok teljesítését. A jó nyomonkövethetőség megkönnyíti az összefoglaló tesztjelentés értelmezését azáltal, hogy feltünteti a tesztbázis elemek állapotát. Ez segíthet a tesztelés technikai vonatkozásainak kommunikálásában az érdekelt felek felé, számukra érthető módon. A nyomonkövethetőség információt biztosít a termék minőségének, a folyamat lehetőségeinek, és a projekt előrehaladásának az üzleti célokkal összevetett kiértékeléséről.

1.4.5 Tesztelési szerepkörök

A tanterv a tesztelésben két fő szerepkört tartalmaz: a tesztmenedzsment szerepkört és a tesztelés szerepkört. A két szerepkörhöz rendelt tevékenységek és feladatok olyan tényezőktől függenek, mint a projekt és a termék környezete, a szerepeket betöltők készségei és maga a szervezet.

A tesztmenedzsment szerepkör átfogó felelősséget vállal a tesztfolyamatért, a tesztcsoportért és a teszttevékenységek vezetéséért. A tesztmenedzsment szerepkör elsősorban a teszttervezési, tesztfelügyeleti és -irányítási, valamint a tesztlezárási tevékenységekre összpontosít. A tesztmenedzsment szerepkör betöltésének módja függ a környezettől. Például agilis szoftverfejlesztésben a tesztmenedzsment feladatok egy részét az agilis csapat átveheti. A több csapatra vagy az egész szervezetre kiterjedő feladatokat a fejlesztői csapaton kívüli tesztmenedzserek is elvégezhetik.

A tesztelés szerepkör átfogó felelősséget vállal a tesztelés tervezési (technikai) vonatkozásaiért. A tesztelés szerepkör elsősorban a tesztelemzés, a műszaki teszttervezés, a tesztmegvalósítás és a tesztvégrehajtás tevékenységére összpontosít.

Különböző időpontokban különböző emberek tölthetik be ezeket a szerepeket. Például a tesztmenedzsment szerepkört elláthatja csoportvezető, a tesztmenedzser, fejlesztési vezető stb. Lehetőség van arra is, hogy egy időben ugyanaz a személy töltse be a tesztelés és a tesztmenedzsment szerepköröket.

1.5 Alapvető készségek és bevált gyakorlatok a teszteléshez

A készség az a képesség, hogy valamit jól csináljunk, ami tudásunkból, gyakorlatunkból és rátermettségünkből fakad. A jó tesztelőknek rendelkezniük kell néhány alapvető képességgel ahhoz, hogy jól végezzék munkájukat. A jó tesztelőknek hatékony csapatjátékosoknak kell lenniük, és képesnek kell lenniük a tesztfüggetlenség különböző szintjein történő tesztelésre.

1.5.1 A teszteléshez szükséges általános készségek

Általánosságban elmondható, hogy a következő készségek különösen fontosak a tesztelők számára:

- Tesztelési ismeretek (a tesztelés hatásosságának növelése érdekében, pl. teszttechnikák használatával)
- Alaposság, körültekintés, kíváncsiság, odafigyelés a részletekre, módszeresség (a hibák azonosítása, különösen a nehezen fellelhetőek esetén)

- Jó kommunikációs készség, aktív hallgatás, csapatjátékos szemlélet (hatásos interakció az összes érdekelt féllel az információ közvetítése, a hibák jelentése és megvitatása, valamint a könnyebb érthetőség érdekében)
- Analitikus gondolkodás, kritikus gondolkodás, kreativitás (a tesztelés hatásosságának növelése érdekében)
- Technikai ismeretek (a tesztelés hatékonyságának növelése érdekében, pl. megfelelő teszteszközök használatával)
- A tevékenységi kör ismerete (a végfelhasználók/üzleti képviselők megértéséhez és a velük való kommunikációhoz)

A tesztelők gyakran rossz hírek hozói. Általános emberi vonás a rossz hír hozóját hibáztatni. Ez döntő fontosságúvá teszi a tesztelők számára a kommunikációs készséget. A teszteredmények közlése a termék és a szerző kritikájaként fogható fel. A megerősítési torzítás megnehezítheti az olyan információk elfogadását, amelyek nem egyeznek a jelenlegi hiedelmekkel. Egyesek a tesztelést pusztító tevékenységként érzékelhetik, pedig nagyban hozzájárul a projekt sikeréhez és a termékminőséghez. Ennek a nézetnek a javítása érdekében a hibákról és meghibásodásokról szóló információkat építő módon kell közölni.

1.5.2 A teljes csapat megközelítés

A tesztelő egyik fontos készsége az, hogy hatékonyan tudjon csapatban dolgozni, és pozitívan tudjon hozzájárulni a csapat céljaihoz. A teljes csapat megközelítés – egy gyakorlat az eXtrém Programozásból (XP) (lásd a 2.1 fejezetet) – erre a készségre épül.

A teljes csapat megközelítésben a csapat bármely tagja, aki rendelkezik a szükséges tudással és képességekkel, bármilyen feladatot elvégezhet, és mindenki felelős a minőségért. A csapattagok ugyanazon a munkaterületen (fizikai vagy virtuális) osztoznak, mivel a közös hely megkönnyíti a kommunikációt és az interakciót. A teljes csapat megközelítés növeli a csapat dinamikáját, javítja a kommunikációt és az együttműködést a csapaton belül, és szinergiát teremt azáltal, hogy lehetővé teszi, hogy a projekt javára fordítsák a csapaton belüli különféle készségkészleteket.

A tesztelők szorosan együttműködnek a csapat többi tagjával, hogy biztosítsák a kívánt minőségi szintek elérését. Ez magában foglalja az üzleti képviselőkkel való együttműködést, azaz, hogy segítsék a megfelelő elfogadási tesztek létrehozását, valamint a fejlesztőkkel való együttműködést, azaz, hogy döntsenek a tesztstratégiáról és a tesztautomatizálási megközelítésekről. A tesztelők így át tudják adni a tesztelési ismereteket a csapat többi tagjának, és befolyásolhatják a termék fejlesztését.

A környezettől függően előfordulhat, hogy a teljes csapat megközelítés nem megfelelő. Bizonyos helyzetekben, mint például a biztonság szempontjából kritikus alkalmazások, magas szintű tesztelői függetlenségre lehet szükség.

1.5.3 A tesztelés függetlensége

Egy bizonyos fokú függetlenség hatékonyabbá teszi a tesztelőt abban, hogy megtalálja a hibákat a szerző és a tesztelő kognitív torzításai közötti különbségek miatt (vö. Salman 1995). A függetlenség azonban nem helyettesíti az ismereteket, például a fejlesztők hatékonyan találhatnak hibákat a saját kódjukban.

A munkatermékeket tesztelheti a szerzőjük (nincs függetlenség), a szerző társai ugyanabból a csapatból (bizonyos függetlenség), a szerzői csapaton kívüli, de a szervezeten belüli tesztelők (nagy függetlenség), vagy a szervezeten kívüli tesztelők (nagyon nagy függetlenség). Általánosságban, a legtöbb projekt esetében az a legjobb, ha a tesztelést több, különböző szintű függetlenséggel végezzük (pl. a fejlesztők komponens-

és komponensintegrációs tesztelést, a tesztcsapat rendszer- és rendszerintegrációs tesztelést, az üzleti képviselők pedig elfogadási tesztelést végeznek).

A tesztelés függetlenségének fő előnye az, hogy a független tesztelők valószínűleg különböző típusú meghibásodásokat és hibákat ismernek fel, mint a fejlesztők, eltérő hátterüknek, technikai szempontjaiknak és előítéleteiknek köszönhetően. Ezenkívül egy független tesztelő ellenőrizheti, megkérdőjelezheti vagy megcáfolhatja az érdekelt felek által a rendszer specifikációja és megvalósítása során megfogalmazott feltételezéseket.

Van azonban néhány hátránya is. A független tesztelők elkülönülhetnek a fejlesztőcsapattól, ami az együttműködés hiányához, kommunikációs problémákhoz vagy a fejlesztői csapattal való ellenséges viszonyhoz vezethet. A fejlesztők elveszíthetik a minőség iránti felelősségérzetüket. A független tesztelőket szűk keresztmetszetnek tekinthetik, vagy hibáztathatják őket a késedelmes kiadásért.

2. Tesztelés a szoftverfejlesztés életciklusán át 130 perc

Kulcsszavak

ellenőrző tesztelés, fehérdoboz tesztelés, feketedoboz tesztelés, elfogadási tesztelés, funkcionális tesztelés, integrációs tesztelés, karbantartási tesztelés, komponensintegrációs tesztelés, komponenstesztelés, nemfunkcionális tesztelés, regressziós tesztelés, rendszerintegrációs tesztelés, rendszertesztelés, shift left, teszt tárgya, tesztszint, teszttípus

Tanulási célok a 2. fejezethez

2.1 Tesztelés a szoftverfejlesztési életciklus kontextusában

AK-2.1.1 (K2) Ismertesse a kiválasztott szoftverfejlesztési életciklus tesztelésre gyakorolt hatását

AK-2.1.2 (K1) Idézzen fel hasznos tesztelési gyakorlatokat, amelyek alkalmazhatóak minden szoftverfejlesztési életciklus esetén

AK-2.1.3 (K1) Idézze fel a test-first megközelítések példáit a fejlesztésben

AK-2.1.4 (K2) Foglalja össze, hogyan hathat a DevOps a tesztelésre

AK-2.1.5 (K2) Ismertesse a shift left megközelítést

AK-2.1.6 (K2) Ismertesse, hogyan használhatóak a visszatekintő megbeszélések a folyamatfejlesztés eszközeiként

2.2 Tesztszintek és teszttípusok

AK-2.2.1 (K2) Különböztesse meg az eltérő tesztszinteket

AK-2.2.2 (K2) Különböztesse meg az eltérő teszttípusokat

AK-2.2.3 (K2) Különböztesse meg az ellenőrző tesztelést és a regressziós tesztelést

2.3 Karbantartási tesztelés

AK-2.3.1 (K2) Foglalja össze a karbantartási tesztelést és az azt szükségessé tevő tényezőket

2.1 Tesztelés egy szoftverfejlesztési életciklus kontextusában

A szoftverfejlesztési életciklusmodell (SDLC) egy absztrakt, magasszintű reprezentációja a szoftverfejlesztési folyamatnak. A szoftverfejlesztési életciklusmodell definiálja, hogy a különböző fejlesztési fázisok és egyéb, a folyamat során végrehajtott tevékenységek hogyan kapcsolódnak egymáshoz logikailag és kronológiailag. Példák a szoftverfejlesztési életciklusmodellekre: szekvenciális fejlesztési modellek (pl.: vízesés modell, V-modell), iteratív fejlesztési modellek (pl.: spirális modell, prototípus fejlesztés) és inkrementális fejlesztési modellek (pl.: Unified Process).

Néhány tevékenység a szoftverfejlesztési folyamatokon belül leírható részletesebb szoftverfejlesztési módszerekkel és agilis gyakorlatokkal is. Például: elfogadásiteszt-vezérelt fejlesztés (angol irodalomban ATDD), viselkedésvezérelt fejlesztés (BDD), szakterület-vezérelt tervezés (DDD), extrém programozás (XP), feature-vezérelt fejlesztés (FDD), Kanban, Lean IT, Scrum és tesztvezérelt fejlesztés (TDD).

2.1.1 A szoftverfejlesztési életciklus tesztelésre gyakorolt hatása

A tesztelést a szoftverfejlesztési életciklus modellhez kell igazítani ahhoz, hogy sikeres legyen. A kiválasztott szoftverfejlesztési életciklus hatással van:

- A tesztelési tevékenységek hatókörére és idejére (pl.: tesztszintek és teszttípusok)
- A tesztelési dokumentációk részletességére
- A teszttechnikák és a tesztmegközelítés kiválasztására
- A tesztautomatizálás mértékére
- A tesztelő szerepkörére és felelősségére

A szekvenciális fejlesztési modellek esetén, a kezdeti fázisok során, a tesztelők tipikusan részt vesznek a követelmények felülvizsgálatában, a tesztelemzésben és a műszaki teszttervezésben. Mivel a futtatható kód általában csak a későbbi fázisokban készül el, ezért a szoftverfejlesztési életciklus korai szakaszában dinamikus tesztelés még nem lehetséges.

Néhány iteratív és inkrementális fejlesztési modell esetében feltételezzük, hogy minden iteráció szállít egy működő prototípust vagy termékinkremenst. Ez azt jelenti, hogy minden iterációban lehetséges mind statikus, mind dinamikus tesztelést végezni az összes tesztszinten. A termékinkremensek gyakori szállítása megköveteli a gyors visszajelzést és a széleskörű regressziós tesztelést.

Az agilis szoftverfejlesztés feltételezi, hogy a projekt minden fázisában történhetnek változások. Következésképpen, az agilis projektekben a nem túl részletes dokumentáció és a széleskörű tesztautomatizálás preferált, annak érdekében, hogy a regressziós tesztelést egyszerűbbé tegyék. Továbbá, a manuális tesztelés jelentős százaléka tapasztalatalapú teszttechnikák (lásd a 4.4. fejezetet) felhasználásával történik, amelyek nem kívánnak széleskörű előzetes tesztelemzést és műszaki teszttervezést.

2.1.2 Szoftverfejlesztési életciklus és jó tesztelési gyakorlatok

Néhány hatékonyan alkalmazható tesztelési gyakorlat a szoftverfejlesztési életciklus modelltől függetlenül:

- Minden szoftverfejlesztési tevékenységhez tartozzon egy teszttevékenység, hogy a minőségellenőrzés minden fejlesztési tevékenységhez biztosítva legyen
- A különböző tesztszintek (lásd a 2.2.1. fejezetet) rendelkezzenek különböző specifikus tesztcélokkal, amelyek lehetővé teszik az átfogó tesztelést a redundancia kiküszöbölésével
- A tesztelemzés és műszaki teszttervezés az adott tesztszinthez tartozó fejlesztési tevékenység során kezdődjön el, így biztosítva a korai tesztelést (lásd a 1.3. fejezetet)

- A tesztelőket be kell vonni a munkatermékek felülvizsgálatába, amint a dokumentációk vázlata elérhető, ezzel is biztosítva, hogy a shift left stratégiát elősegíti a korai tesztelés és hibafelismerés (lásd a 2.1.5. fejezetet)

2.1.3 A tesztelés, mint a szoftverfejlesztés hajtóereje

A TDD, ATDD és BDD hasonló fejlesztési megközelítések, ahol a tesztek meghatározása a fejlesztés irányításának eszközeként szolgál. Mindegyik megközelítés megvalósítja a korai tesztelés elvét (lásd a 1.3. fejezetet) és követi a shift left megközelítést (lásd: 2.1.5-ös fejezet), mivel a tesztek meghatározása előbb történik, mint a kód megírásra. Ezek a megközelítések az iteratív fejlesztési modellt támogatják és az alábbiak szerint lehet jellemezni őket:

Tesztvezérelt fejlesztés (TDD):

- A kódolást tesztesetek irányítják (az átfogó szoftvertervek helyett) (Beck 2003)
- Először a tesztek kerülnek megírásra, majd azt követően a kód, olyan módon, hogy a teszteknek megfeleljen. Végül a teszteket és a kódot is refaktorálják.

Elfogadásiteszt-vezérelt fejlesztés (ATDD) (lásd a 4.5.3. fejezetet):

- A teszteket elfogadási feltételekből származtatja a rendszertervezési folyamat részeként (Gärtner 2011)
- A teszteket az előtt írják meg, hogy lefejesztésre kerülne az alkalmazás azon része, mely kielégíti ezeket a teszteket.

Viselkedésvezérelt fejlesztés (BDD)

- Kifejezi az alkalmazás kívánt viselkedését olyan tesztesetekkel, melyek egyszerű formában, természetes nyelven vannak megírva, és így könnyen érthető az érdekelt felek számára – általában a "Given/When/Then" formát használva (Chelimsky 2010)
- A teszteseteket ezután automatikusan végrehajtható tesztekké alakítják.

A fent említett megközelítések alkalmazása során, a tesztek megmaradhatnak automatizált tesztek formájában, hogy biztosítsák a kód minőségét jövőbeli módosítások esetén.

2.1.4 DevOps és tesztelés

DevOps egy szervezeti megközelítés, melynek célja a szinergia létrehozása a fejlesztés (beleértve a tesztelést) és az üzemeltetés között, hogy közös célokat érjenek el. A DevOps egy kulturális változást igényel a szervezeten belül, hogy áthidalja a fejlesztés (beleértve a tesztelést) és az üzemeltetés közötti réseket, miközben azok szerepét egyenértékűnek tekinti. A DevOps támogatja a csapat önállóságát, a gyors visszajelzést, az integrált eszközrendszerrel való munkát, valamint olyan technikai gyakorlatokat, mint a folyamatos integráció (CI) és a folyamatos szállítás (CD). Ez lehetővé teszi a csapatok számára, hogy a DevOps szállítási folyamatláncon (delivery pipeline) keresztül gyorsabban készítsenek, teszteljenek és adjanak ki magas minőségű kódot (Kim 2016).

A tesztelés szempontjából nézve a DevOps néhány előnye:

- Gyors visszajelzés a kód minőségéről és arról, hogy a változtatások hogyan befolyásolják a már meglévő kódot
- A folyamatos integráció elősegíti a shift left megközelítést a tesztelésben (lásd a 2.1.5. fejezetet), mivel bátorítja a fejlesztőket, hogy jó minőségű kódot adjanak át, komponenstesztekkel és statikus elemzéssel kísérve
- Támogatja az automatizált folyamatokat, mint például a folyamatos integrációt/szállítást, amelyek segítik a stabil tesztkörnyezetek kialakítását

- Növeli a fókuszt a nemfunkcionális jellemzőkön (pl. teljesítmény, megbízhatóság)
- Az automatizálás a szállítási folyamatláncon keresztül csökkenti az ismétlődő manuális tesztek szükségességét
- Minimális a regresszió okozta kockázat az automatizált regressziós tesztek méretének és hatókörének köszönhetően

A DevOps sem mentes a kockázatoktól és kihívásoktól, például:

- A DevOps szállítási folyamatláncot (delivery pipeline) meg kell tervezni és meg kell valósítani
- A folyamatos integrációs/szállítási eszközöket be kell vezetni és karban kell tartani
- A tesztautomatizálás további erőforrásokat igényel, illetve nehéz lehet kialakítani és karbantartani

Bár a DevOps megköveteli egy magasabb fokú automatizált tesztelés megvalósítását, a manuális tesztelés – különösen a felhasználó szempontjából – továbbra is szükséges.

2.1.5 A shift left megközelítés

Az korai tesztelés elve (lásd a 1.3. fejezetet) néha shift left néven is említésre kerül. A shift left esetén az a cél, hogy a szoftverfejlesztés életciklusának folyamatában a tesztelés minél korábban legyen elvégezve (pl. nem várni a kód megvalósítására vagy a komponensek integrálására). Viszont ez nem jelenti azt, hogy a szoftverfejlesztés életciklusának későbbi szakaszaiban a tesztelés elhanyagolható lenne.

A tesztelésben a shift left megvalósítását elősegítő számos jó gyakorlat létezik, mint például:

- Specifikációk felülvizsgálata a tesztelés szempontjából. Ezek a felülvizsgálatok gyakran fedeznek fel lehetséges hibákat, amik például félreértésekből, hiányosságokból vagy ellentmondásokból származnak
- Tesztesetek megírása a kód létrejötte előtt, majd a kód futtatása egy teszttámogató szoftverkörnyezetben a kód implementálása közben
- Folyamatos integráció és még jobb esetben folyamatos szállítás alkalmazása, mivel ezek gyors visszajelzést adnak, illetve automatizált komponensteszteket biztosítanak a kód mellé, amikor azok tárolóba (repository) kerülnek.
- A forráskód statikus elemzése a dinamikus tesztelés előtt, vagy egy automatizált folyamat részeként
- Nemfunkcionális tesztelés megvalósítása, lehetőség szerint komponensteszt-szinten kezdve. Ez egy formája a shift left megvalósításának, mert a nemfunkcionális teszttípus általában a szoftverfejlesztési életciklus későbbi részeiben történik, mikor a teljes rendszer és egy reprezentatív tesztkörnyezet már a rendelkezésünkre áll.

A shift left megközelítés extra képzést, költségeket eredményezhet a folyamat korai szakaszaiban, de várható, hogy a későbbiekben megtakarításhoz vezet.

A shift left megközelítéshez fontos, hogy az érdekelt felek elkötelezettek legyenek a koncepció irányában.

2.1.6 Visszatekintő megbeszélések és folyamatfejlesztés

A visszatekintő megbeszéléseket (projekt retrospektív) gyakran a projekt vagy egy iteráció végén, vagy egy kiadási mérföldkőnél tartják, de igény szerint bármikor tarthatóak. A visszatekintő megbeszélések időzítése és lebonyolítása a követett szoftverfejlesztési életciklustól függ. Ezeken a megbeszéléseken a résztvevők (nem csak tesztelők, hanem például fejlesztők, szoftver architektek, terméktulajdonosok, üzleti elemzők is) megvitatják:

- Mi volt sikeres, és mit kellene megtartani?

- Mi nem volt sikeres, és mit lehetne fejleszteni?
- Hogyan lehet beépíteni az újításokat és megtartani a sikereket a jövőben?

Az eredményeket rögzíteni kell és általában részei az összefoglaló tesztjelentésnek (lásd az 5.3.2. fejezetet). A visszatekintő megbeszélések kiemelten fontosak a folyamatos fejlesztés sikeres megvalósítása, és a javasolt fejlesztések nyomonkövethetősége miatt.

Tipikus előnyök tesztelési szempontból:

- Növelt teszthatékonyság/hatásosság (pl. folyamatfejlesztési javaslatok végrehajtásával)
- A tesztver megnövekedett minősége (pl. a tesztfolyamatok közös felülvizsgálata által)
- Csapatösszekovácsolódás és tanulás (pl. azáltal, hogy lehetőség nyílik problémák megfogalmazására és javaslatok megtételére)
- A tesztbázis jobb minősége (pl. rá lehet mutatni a követelmények terjedelembeli, illetve minőségbeli hiányosságaira, és meg lehet oldani ezeket)
- Jobb együttműködés a fejlesztés és a tesztelés között (pl. a közös munka rendszeres felülvizsgálata és optimalizálása által)

2.2 Tesztszintek és teszttípusok

A tesztszintek teszttevékenységek olyan csoportjai, amelyeket együtt szerveznek és kezelnek. Minden tesztszint a tesztfolyamat egy példánya, a szoftverfejlesztési folyamat adott pontján kerül végrehajtásra, a komponensektől kezdve a teljes rendszerekig, illetve adott esetben a rendszerek rendszeréig.

A tesztszintek a szoftverfejlesztési életcikluson belüli egyéb tevékenységekhez kapcsolódnak. A szekvenciális szoftverfejlesztési életciklusok esetén a tesztszintek gyakran úgy vannak meghatározva, hogy az egyik szint kilépési feltételei a következő szint belépési feltételeinek a részét képezik. Néhány iteratív modell esetén ez nem feltétlenül érvényes. A fejlesztési tevékenységek több tesztszinten is átívelhetnek. A tesztszintek időben átfedhetik egymást.

A teszttípusok teszttevékenységek olyan csoportjai, amelyek specifikus minőségjellemzőkhöz kapcsolódnak. Ezeknek a tevékenységeknek a többségét minden tesztszinten el lehet végezni.

2.2.1 Tesztszintek

Ebben a tantervben a következő öt tesztszint kerül ismertetésre:

- **Komponenstesztelés** (más néven egységtesztelés): az egyes komponensek elkülönítve történő tesztelésére összpontosít. Gyakran speciális támogatást igényel, például teszttámogató szoftverkörnyezetet vagy egységteszt-keretrendszert. A komponenstesztelést általában a fejlesztők végzik a saját fejlesztési környezetükben.
- **Komponensintegrációs tesztelés** (más néven egységintegrációs tesztelés): a komponensek közötti interfészek és interakciók tesztelésére összpontosít. Erősen függ az integrációs stratégia megközelítésétől, mint a lentről-fölfelé, föntről-lefelé vagy nagy-bumm.
- **Rendszertesztelés:** a teljes rendszer vagy termék viselkedésére és képességeire összpontosít, gyakran magában foglalva a végponttól végpontig tartó feladatok funkcionális tesztelését és a minőségjellemzők nemfunkcionális tesztelését. Néhány nemfunkcionális minőségjellemző esetében előnyösebb teljes rendszeren, egy reprezentatív tesztkörnyezetben tesztelni (például felhasználhatóság esetében). Alrendszerek szimulációjának használata is lehetséges. A rendszer tesztelését független tesztcsoport is végezheti, és a rendszer specifikációihoz kapcsolódik.

- **Rendszerintegrációs tesztelés:** a tesztelt rendszer, egyéb rendszerek és külső szolgáltatások interfészeinek tesztelésére összpontosít. A rendszerintegrációs tesztelés megfelelő tesztkörnyezeteket kíván meg, lehetőleg minél hasonlóbbat az üzemeltetési környezethez.
- **Elfogadási tesztelés:** validálásra és a kiadásra való felkészültség demonstrálására összpontosít, más szavakkal, azt vizsgáljuk, hogy a rendszer valóban kielégíti-e a felhasználó üzleti igényeit. Ideális esetben az elfogadási tesztelést a tervezett felhasználók végzik. Fő formái: felhasználói elfogadási tesztelés (angolul user acceptance testing, UAT), működési elfogadási tesztelés, szerződéses és szabályozói elfogadási tesztelés, alfatesztelés és bétatesztelés.

A tesztszintek megkülönböztethetőek a következő, nem kimerítő paraméterek alapján, hogy elkerülhető legyen a teszttevékenységek átfedése:

- Teszt tárgya
- Tesztcélok
- Tesztbázis
- Hibák és meghibásodások
- Megközelítés és felelősségek

2.2.2 Teszttípusok

Számos teszttípus létezik és alkalmazható egy projektben. Ebben a tantervben a következő négy teszttípust említjük meg:

Funkcionális tesztelés során a komponens vagy rendszer működését vizsgáljuk, abból a szempontból, hogy elvégzi-e a meghatározott funkcióit. A meghatározott funkciók azt fejezik ki, hogy „mit” kell tudnia a rendszernek elvégeznie. A fő céljai a funkcionális tesztelésnek: a funkcionális teljesség, a funkcionális helyesség és a funkcionális megfelelőség ellenőrzése.

Nemfunkcionális tesztelés során olyan tulajdonságokat ellenőrzünk, amelyek nem a komponens vagy rendszer funkcionális jellemzői. A nemfunkcionális tesztelés azt vizsgálja, hogy „miként" viselkedik a rendszer. A fő célja a nemfunkcionális szoftver-minőségjellemzők ellenőrzése. Az ISO/IEC 25010 szabvány a következő osztályozást adja a nemfunkcionális szoftver-minőségjellemzőknek:

- Teljesítményhatékonyság
- Kompatibilitás
- Használhatóság
- Megbízhatóság
- Biztonság
- Karbantarthatóság
- Hordozhatóság

Előfordulhat, hogy a nemfunkcionális tesztelés az életciklus korai szakaszában kezdődik (például a felülvizsgálatok, a komponenstesztelés vagy a rendszertesztelés részeként). Számos nemfunkcionális teszt származtatható funkcionális tesztekből, mivel egy már létező funkcionális teszt futtatása során ellenőrizhetünk nemfunkcionális jellemzőket is (például ellenőrizhetjük, hogy egy funkció végrehajtásához szükséges idő megfelelő-e, vagy végrehajtódik-e egy eltérő keretrendszeren belül). A későn megtalált nemfunkcionális hibák nagyban befolyásolhatják egy projekt sikerét. Nemfunkcionális tesztelés esetén néha szükségünk lehet specifikus tesztkörnyezetre, mint például használhatósági teszt esetén egy használhatósági laborra.

A **feketedoboz tesztelés** (lásd a 4.2. fejezetet) egy specifikációalapú teszttípus, ahol a tesztelés tárgyától különálló dokumentációkból származtatjuk a teszteseteket. A feketedoboz tesztelés fő célja a rendszer viselkedésének ellenőrzése a rendszer specifikációinak szempontjából.

A **fehérdoboz tesztelés** (lásd a 4.3. fejezetet) egy struktúraalapú teszttípus, ahol a rendszer belső szerkezete vagy az implementációja (például: kód, architektúra, munkafolyamat, adatfolyam) alapján származtatjuk a teszteseteket. A fehérdoboz tesztelés fő célja, hogy tesztekkel elfogadható szinten lefedjük a rendszer alapvető struktúráját.

Mind a négy teszttípus alkalmazható minden tesztszinten, bár a fókusz minden szinten más lesz. Számos különböző teszttechnika használható a tesztfeltételek és tesztesetek származtatására minden teszttípus esetében.

2.2.3 Ellenőrző tesztelés és regressziós tesztelés

Változásokat tipikusan akkor eszközölünk egy komponensen vagy rendszeren, mikor új funkcionalitásokat szeretnénk bevezetni vagy hibákat szeretnénk javítani. A tesztelésnek ebben az esetben ellenőrző és regressziós tesztelést is tartalmaznia kell.

Az **ellenőrző tesztelés** megerősíti, hogy az adott hiba sikeresen javítva lett. A kockázattól függően, a javított szoftververziót több módon is tesztelhetjük:

- újrafuttatva az összes tesztesetet, amelyek korábban elbuktak az adott hiba miatt, vagy
- kiegészítve a tesztfuttatást új tesztekkel, amelyek lefedik a javítás által bevezetett módosításokat is

Azonban, mikor hibajavítás során a rendelkezésre álló erőforrások (idő és pénz) nem elegendőek, az ellenőrző tesztelés korlátozódhat csak a meghibásodást előhozó lépések megismétlésére, és annak ellenőrzésére, hogy a meghibásodás nem történik meg.

A **regressziós tesztelés** megerősíti, hogy egy változtatásnak nincsenek kedvezőtlen következményei. A változtatások közé értjük a javításokat is, amelyek már átestek az ellenőrző tesztelésen. Ezek a kedvezőtlen következmények hatással lehetnek az adott komponensre, ahol a javítást végeztük, más komponensekre a rendszeren belül vagy akár más kapcsolódó rendszerekre. A regressziós tesztelés nem feltétlenül korlátozódik magára a teszt tárgyára, lehetséges, hogy a tesztkörnyezet változása miatt szükséges a tesztelés. Annak érdekében, hogy optimalizálni lehessen a szükséges regressziós tesztelés mértékét, javasolt először egy hatáselemzés elvégzése. A hatáselemzés megmutatja, hogy a szoftver mely részei lehetnek érintettek a módosítás által.

A regressziós tesztkészletek számos alkalommal futnak és általánosságban a regressziós tesztek száma növekedik minden iterációval vagy kiadással, ezért remek választás ezeket a teszteket automatizálni. Ezen tesztek automatizációját a projekt korai szakaszában ajánlatos elkezdeni. Ahol folyamatos integrációt használnak, például DevOps esetén (lásd a 2.1.4. fejezetet), ott jó gyakorlat az automatizált regressziós tesztek alkalmazása. Ezért, a szituációtól függően, regressziós tesztelés akár különböző tesztszinteken is alkalmazható.

Az ellenőrző tesztelés és/vagy a regressziós tesztelés szükséges minden tesztszinten, ha hibák kerülnek kijavításra és/vagy változtatások történnek ezeken a tesztszinteken.

2.3 Karbantartási tesztelés

A karbantartásnak különböző kategóriái vannak, lehet javító, a változó környezethez alkalmazkodó, illetve fejlesztheti a teljesítményt vagy a karbantarthatóságot (részletekért lásd: ISO/IEC 14764). A karbantartás része lehet a tervezett kiadásoknak vagy a nem tervezett javításoknak (hot fixes) is. Hatáselemzést érdemes lehet végezni mielőtt bármilyen változtatást eszközölünk, hogy a döntésünket a változással kapcsolatban annak fényében tudjuk meghozni, hogy milyen lehetséges következményei lehetnek a rendszer egyéb területeire. Egy használatban lévő rendszerben történő változások tesztelése során két szempontot kell

figyelembe vennünk: ki kell értékelnünk, hogy sikeres volt-e a változás megvalósítása, és ellenőriznünk kell a lehetséges regressziókat a rendszer változatlanul maradt részében (mely általában a rendszer nagyobb része).

A karbantartási tesztelés általában a következő tényezőktől függ:

- A változtatás kockázatának mértéke
- A meglévő rendszer mérete
- A változtatás mérete

A karbantartás és a karbantartási tesztelés kiváltó okai pedig a következőképpen oszthatók fel:

- Módosítások, például tervezett fejlesztések (kiadásalapú), hibajavítások vagy hot fixek.
- Frissítések vagy migrációk az üzemeltetési környezetben, például egyik platformról a másikra való áttérés során. Ez új környezettel kapcsolatos teszteket, valamint a megváltozott szoftverrel kapcsolatos teszteket igényelhet. Adatkonverziós tesztekre is szükség lehet, ha egy másik alkalmazásból származó adatokat migrálnak a karbantartott rendszerbe.
- Visszavonultatás, például amikor egy alkalmazás elérte élettartamának végét. Egy rendszer visszavonultatása adatarchiválási teszteket is igényelhet, ha a visszavonulást követően hosszú adattárolási időszakokra van szükség. Az archiválás utáni visszaállítási és visszahozatali eljárások tesztelése is szükségessé válhat abban az esetben, ha a tárolási időszak alatt bizonyos adatokra szükség van.

3. Statikus tesztelés 80 perc

Kulcsszavak

anomália, átvizsgálás, dinamikus tesztelés, felülvizsgálat, formális felülvizsgálat, informális felülvizsgálat, inspekció, statikus elemzés, statikus tesztelés, technikai felülvizsgálat

Tanulási célok a 3. fejezethez

3.1. A statikus tesztelés alapjai

AK-3.1.1. (K1) Ismerje fel azokat a munkatermék típusokat, amelyek vizsgálhatók a különböző statikus teszttechnikákkal

AK-3.1.2. (K2) Ismertesse a statikus tesztelés értékeit

AK-3.1.3. (K2) Hasonlítsa össze a statikus és a dinamikus tesztelést és vizsgálja meg a különbségeket

3.2. A visszajelzés és a felülvizsgálat folyamata

AK-3.2.1. (K1) Azonosítsa az érdekelt felek számára adott korai és gyakori visszajelzések előnyeit

AK-3.2.2. (K2) Foglalja össze a felülvizsgálati folyamat egyes tevékenységeit

AK-3.2.3. (K1) Idézze fel, hogy milyen felelősségek rendelhetők a főbb szerepekhez a felülvizsgálatok végrehajtása során

AK-3.2.4. (K2) Hasonlítsa össze a különböző felülvizsgálati típusokat és vizsgálja meg a különbségeket

AK-3.2.5. (K1) Idézze fel a sikeres felülvizsgálathoz hozzájáruló tényezőket

3.1 A statikus tesztelés alapjai

A dinamikus teszteléssel ellentétben, statikus tesztelés esetén a tesztelés alatt álló szoftver futtatása nem szükséges. Kódot, folyamat specifikációt, rendszer felépítés specifikációt vagy más munkatermékeket értékelünk manuális vizsgálatok során (például felülvizsgálatokkal) vagy eszköz segítségével (például statikus elemzés során). Tesztcél lehet a minőség javítása, a hibák felderítése, vagy például olyan jellemzők vizsgálata, mint olvashatóság, a teljesség, a helyesség, tesztelhetőség és a konzisztencia. Statikus tesztelést alkalmazhatunk mind verifikálás, mind validálás során.

Tesztelők, az üzleti oldal képviselői és a fejlesztők együtt dolgoznak a mintaként szolgáló leképezések kialakításában, az együttműködés során kialakított felhasználói történetek megírása során, valamint a teendőlista finomítására szolgáló megbeszélések során annak biztosítására, hogy a felhasználói történetek és a hozzájuk kapcsolt munkatermékek megfeleljenek a meghatározott követelményeknek, mint például a Definition of Ready fogalmának (lásd az 5.1.3. fejezetet). A felülvizsgálati technikák alkalmazhatók arra, hogy biztosítsuk a felhasználói történetek érthetőségét és teljességét, valamint biztosítják, hogy tartalmaznak tesztelhető elfogadási feltételeket is. A megfelelő kérdéseket felvetve a tesztelők feltárják, megkérdőjelezik és segítik javítani a javasolt felhasználói történeteket.

A statikus elemzés még a dinamikus tesztelés előtt feltárhat problémákat, miközben gyakran kevesebb erőfeszítést igényel, hiszen nincs szükség tesztesetekre, és jellemzően eszközöket is használnak hozzá (lásd a 6 fejezetet). A statikus elemzést gyakran beépítik a folyamatos integrációs keretrendszerekbe (lásd a 2.1.4 fejezetet). Míg nagyrészt arra használják, hogy specifikus kódhibákat találjanak a segítségével, a statikus elemzést arra is használják, hogy értékeljék a karbantarthatóságot és a biztonságot. A helyesírás ellenőrzők és az olvashatóságot elemző eszközök további példákként szolgálnak a statikus elemzési eszközökre.

3.1.1 Statikus teszteléssel vizsgálható munkatermékek

Csaknem bármilyen munkatermék vizsgálható statikus tesztelés alkalmazásával. Példaként említhetők a követelmény specifikáció dokumentumai, forráskód, teszttervek, tesztesetek, teendőlista elemek, tesztvázlatok, projekt dokumentumok, szerződések és modellek.

Bármely munkatermék, amely olvasható és megérthető, lehet felülvizsgálat tárgya. Azonban statikus elemzéshez fontos, hogy a munkaterméknek legyen szerkezete, aminek alapján ellenőrizhető (például modellek, kód, vagy bármely szöveg, ami formális szintaktikával rendelkezik).

Statikus tesztelésre nem alkalmas munkatermékeknek tekinthetők azok, amelyek emberek számára nehezen értelmezhetők, és amelyeket nem szabad elemeznünk eszközök segítségével sem (mint például a harmadik fél által biztosított végrehajtható kód, ahol jogi kérdések merülhetnek fel).

3.1.2 A statikus tesztelés értéke

A statikus tesztelés a szoftverfejlesztési életciklus legkorábbi szakaszaiban is képes hibákat kimutatni, ezzel megvalósítva a korai tesztelés elvét (lásd az 1.3. fejezetet). Olyan hibákat is képes azonosítani, amelyek dinamikus teszteléssel nem deríthetők fel (mint például: elérhetetlen kód; tervezési mintákat, amelyek nem a kívántaknak megfelelően lettek megvalósítva; hibák a végre nem hajtható munkatermékekben).

A statikus tesztelés segítségével képessé válunk, hogy értékeljük a munkatermékek minőségét, és bizalmat építsünk ki ezekkel szemben. A dokumentált követelmények verifikálásával az érdekelt felek biztosak lehetnek abban, hogy ezek a követelmények az aktuális igényeket írják le. Mivel a statikus tesztelés a szoftverfejlesztési életciklus korai szakaszában végrehajtható, kölcsönös megértés hozható létre, és a kommunikáció is javítható az érintett érdekelt felek között. Ebből következik, hogy ajánlott az érdekelt felek széles körének részvétele a statikus tesztelésben.

Bár a felülvizsgálatok megvalósítása költséges lehet, a teljes projektköltség általában sokkal alacsonyabb, mint amikor nem hajtanak végre felülvizsgálatokat, mert kevesebb időre és erőfeszítésre van szükség a hibák kijavítására a projekt későbbi szakaszaiban.

Statikus elemzés használatával hatékonyabban tudunk hibákat kimutatni, mint a dinamikus tesztelés során, így ez rendszerint kevesebb kódhibát eredményez, és ez alacsonyabb általános fejlesztési erőfeszítést igényel.

3.1.3 Statikus és dinamikus tesztelés közötti különbségek

A statikus és a dinamikus tesztelés kiegészíti egymást. Hasonló céljaik vannak, mert mindkettő támogatja a hibák kimutatását a munkatermékekben (lásd az 1.1.1. fejezetet), de eltérések is vannak közöttük, mint például:

- A statikus és a dinamikus tesztelés (a meghibásodások elemzésével) egyaránt a hibák megtalálására vezet, azonban van néhány hibatípus, amelyet csak statikus vagy dinamikus teszteléssel lehet megtalálni.
- A statikus tesztelés közvetlenül a hibákat találja meg, míg a dinamikus tesztelés meghibásodásokat tár fel, amelyekből a csatolt hibák az ezt követő elemzés során határozhatók meg.
- A statikus tesztelés könnyebben kimutathat olyan hibákat, amelyek elhelyezkedése a kódban a ritkábban bejárt végrehajtási útvonalakon találhatók, vagy olyanokat, amelyek nehezen érhetők el dinamikus teszteléssel.
- Statikus tesztelés alkalmazható végre nem hajtható munkatermékek esetén, míg dinamikus tesztelés csak futtatható munkatermékek esetén alkalmazható.
- Statikus tesztelés felhasználható azon minőségjellemzők mérésére, amelyek a futtatható kódtól függetlenek (mint például karbantarthatóság), míg a dinamikus tesztelés felhasználható olyan minőségjellemzők mérésére, amelyek a kód végrehajtástól függenek (mint például a teljesítményhatékonyság).

A statikus tesztelés során könnyebben, és/vagy kevesebb ráfordítással megtalálható tipikus hibák közé tartoznak az alábbiak:

- Követelmény hibák (mint például inkonzisztenciák, kétértelműségek, ellentmondások, kihagyások, pontatlanságok, duplikációk)
- Tervezési hibák (mint például nem kellően hatékony adatbázis szerkezetek, szegényes modularizáció)
- Bizonyos típusú kódolási hibák (mint például meghatározatlan értékeket tartalmazó változók, létre nem hozott változók, elérhetetlen vagy duplikált kód, felesleges kód komplexitás)
- Eltérések a szabványoktól (mint például az elnevezési konvenciók be nem tartása a kódolási szabványoknál)
- Helytelen interfész specifikációk (mint például nem illeszkedő paraméter szám, típus, vagy sorrend)
- A biztonsági sebezhetőségek bizonyos típusai (mint például a puffer túlcsordulás)
- Hiányok vagy pontatlanságok a tesztbázis lefedettségben (mint például hiányzó tesztek egy adott elfogadási feltételre)

3.2 A visszajelzés és a felülvizsgálat folyamata

3.2.1 Az érdekelt felek korai és gyakori visszajelzésének előnyei

A korai és gyakori visszajelzés lehetőséget biztosít a potenciális minőségi problémák korai kommunikációjára. Amennyiben kismértékű az érdekelt felek részvétele a szoftverfejlesztési életciklus során, a fejlesztés alatt álló termék lehet, hogy nem felel meg az érdekelt felek eredeti vagy aktuális elképzelésének. Amennyiben nem sikerül kielégíteni az érdekelt felek termékkel kapcsolatos igényeit, ez költséges átdolgozáshoz, elmulasztott határidőkhöz, egymásra mutogatáshoz vezethet, és a projekt teljes kudarca is előfordulhat.

Az érdekelt felek gyakori visszajelzése a szoftverfejlesztési életciklus során megakadályozhatja a követelmények félreértését, és biztosíthatja, hogy a követelményekben bekövetkező változások érthetők legyenek, és hamar megvalósításra kerüljenek. Ez segíti a fejlesztő csapatot, hogy jobban megértsék, mit is készítenek. Lehetővé teszi számukra, hogy azokra a funkciókra fókuszáljanak, amelyek az érdekelt felek számára a legfontosabbak, és a leginkább pozitív befolyásuk van a beazonosított kockázatokra.

3.2.2 A felülvizsgálati folyamat tevékenységei

Az ISO/IEC 20246 szabvány meghatároz egy általános felülvizsgálati folyamatot, amely struktúrált, de flexibilis keretrendszert biztosít, amely alapján testre szabható adott helyzetre egy specifikus felülvizsgálati folyamat. Amennyiben a szükséges felülvizsgálat formálisabb, akkor több, a különböző tevékenységekre meghatározott feladatra lesz szükség a folyamatban.

Sok munkatermék mérete olyan nagynak bizonyulhat, hogy egyetlen felülvizsgálattal nem fedhető le. A felülvizsgálati folyamat több alkalommal indítható, hogy a felülvizsgálatot a teljes munkatermékre el tudjuk végezni.

A felülvizsgálati folyamat tevékenységei a következők:

- **Tervezés**. A tervezési fázis során meg kell határozni a felülvizsgálat hatókörét, amely magába foglalja a felülvizsgálat célját, meg kell határozni a felülvizsgálatra bocsátandó munkaterméket, az értékelendő minőségjellemzőket, a megcélzott területeket, a kilépési feltételeket, a felülvizsgálatot alátámasztó információkat, mint például a szabványok, a felülvizsgálat időkeretét és ráfordításait.
- **Felülvizsgálat indítása**. A felülvizsgálat megindítása során a cél az, hogy biztosítsuk, minden és mindenki legyen készen a felülvizsgálat megkezdésére. Ez magába foglalja azt is, hogy minden egyes résztvevőnek legyen hozzáférése a vizsgálat alá vont munkatermékhez, mindenki tisztában van a szerepével és a felelősségeivel és megkap mindent, ami a felülvizsgálat végrehajtásához szükséges.
- **Egyéni felülvizsgálat**. Minden felülvizsgáló egyedi felülvizsgálatot végez a felülvizsgálat alá vont munkatermék minőségének értékelésére, továbbá anomáliákat, javaslatokat és kérdéseket azonosít egy vagy több felülvizsgálati technika alkalmazásával (mint például ellenőrzőlista-alapú felülvizsgálat, forgatókönyv-alapú felülvizsgálat). Az ISO/IEC 20246 szabvány további részleteket tárgyal a különböző felülvizsgálati technikákkal kapcsolatban. A felülvizsgálók feljegyzik az összes beazonosított anomáliát, javaslatot és kérdést.
- **Kommunikáció és elemzés**. Mivel a felülvizsgálat során azonosított anomáliák nem szükségszerűen mind hibák, az összes ilyen anomáliát elemezni és megvitatni szükséges. Minden egyes anomália esetén döntést kell hozni annak állapotáról, hovatartozásáról, és a szükséges tennivalókról. Ezt tipikus esetben a felülvizsgálati értekezlet során végezzük el, ahol a résztvevők arról is döntenek, hogy milyen a minőségi szintje a felülvizsgálat alá vont munkaterméknek, és milyen követő tevékenységek szükségesek. Szükség lehet egy követő felülvizsgálatra is a tevékenységek lezárásához.

- **Javítás és jelentés**. Minden egyes hiba esetében hibajelentést kell létrehozni, hogy a javítási tevékenységek követhetők legyenek. Amennyiben teljesítettük a kilépési feltételeket, a munkatermék elfogadható. A felülvizsgálat eredményeit jelentésben foglaljuk össze.

3.2.3 Szerepek és felelősségek felülvizsgálat esetében

A felülvizsgálatokon különböző érdekelt felek vesznek részt, akik számos szerepet tölthetnek be. A főbb szerepek és a hozzájuk tartozó felelősségek a következők:

- Menedzser - meghatározza, hogy mit kell felülvizsgálni, és erőforrásokat biztosít, mint például személyeket és időkeretet a felülvizsgálat végrehajtásához
- Szerző - létrehozza és javítja a felülvizsgálat alá vett munkaterméket
- Moderátor (témavezetőként is ismert) - biztosítja a felülvizsgálati értekezletek hatékony levezetését, beleértve a mediálást, idő menedzsmentet, és egy biztonságos felülvizsgálati környezetet, ahol mindenki szabadon hozzászólhat
- Jegyzőkönyvvezető (írnokként is ismert) - összegyűjti az anomáliákat a felülvizsgálóktól és feljegyzi a felülvizsgálati információkat, mint például a döntéseket és a felülvizsgálati értekezlet során talált új anomáliákat
- Felülvizsgáló - végrehajtja a felülvizsgálatokat. Felülvizsgáló lehet olyan személy, aki a projekten dolgozik, a téma szakértője, vagy bármely más érdekelt fél
- Felülvizsgálat-vezető - általános felelősséget vállal a felülvizsgálatban, mint például döntéseket hoz, ki vegyen részt a felülvizsgálatban, és megszervezi, mikor és hol folytatják le a felülvizsgálatot.

További, jobban részletezett szerepek is lehetségesek, ahogy ez leírásra került az ISO/IEC 20246 szabványban.

3.2.4 Felülvizsgálat típusok

Számos felülvizsgálati típus létezik, amelyek az informális felülvizsgálattól a formális felülvizsgálatokig terjednek. A megkövetelt formalitási szint olyan faktoroktól függ, mint például a követett szoftverfejlesztési életciklus, a fejlesztési folyamat érettsége, a felülvizsgálat alá vont munkatermék fontossága és komplexitása, jogi vagy szabályozói követelmények, és az auditálás szükségessége. Ugyanazon munkatermék felülvizsgálható különböző felülvizsgálati típusok segítségével, mint például kezdetben egy informális típus alkalmazásával és később egy formálisabb típus használatával.

A megfelelő felülvizsgálati típus kiválasztása kulcsszerepet játszik abban, hogy elérjük az elvárt felülvizsgálati célokat (lásd a 3.2.5 fejezetetet). A kiválasztás nem csak a célokon alapul, hanem olyan faktoroktól is függ, mint a projekt szükségletei, rendelkezésre álló erőforrások, a munkatermék típusa és a kapcsolódó kockázatok, a szakterület és a céges kultúra.

Néhány általánosan használt felülvizsgálati típus a következő:

- **Informális felülvizsgálat.** Az informális felülvizsgálatok nem követnek előre meghatározott folyamatot, és nem igényelnek formális, dokumentált kimenetet. A fő cél az anomáliák kimutatása.
- **Átvizsgálás.** Egy átvizsgálás, amelyet a szerző vezet, számos célt szolgálhat ki, mint például a munkatermék minőségének értékelése, és bizalom kiépítése a munkatermékkel szemben, a felülvizsgálók képzése és oktatása, az egyetértés megteremtése, új ötletek generálása, a szerzők motiválása és segítése a jobb munkavégzésre és az anomáliák felismerésére. A felülvizsgálók végrehajthatnak egyéni felülvizsgálatot az átvizsgálást megelőzően, de ez nem kötelező.

- **Technikai felülvizsgálat.** Egy technikai felülvizsgálatot szakmailag képzett felülvizsgálók végeznek, és a felülvizsgálatot moderátor vezeti. Egy technikai felülvizsgálat céljai lehetnek, hogy egyetértésre jussunk és döntéseket hozzunk egy adott technikai problémát illetően, ugyanakkor cél lehet anomáliák kimutatása, minőség értékelése, bizalom kiépítése a munkatermékkel szemben, új ötletek generálása, a szerzők motiválása és segítése a jobb munkavégzésre.
- **Inspekció.** Mivel az inspekció a leginkább formális típusú felülvizsgálat, ezért teljes általános folyamatot követi (lásd a 3.2.2. fejezetet). A fő cél az, hogy megtaláljuk a legtöbb anomáliát. További célok lehetnek a minőség értékelése, bizalom kiépítése a munkatermékkel szemben, a szerzők motiválása és segítése a jobb munkavégzésre. Metrikákat gyűjtünk, és ezek felhasználhatók a szoftverfejlesztési életciklus javítására, beleértve az inspekciós folyamat javítását. Inspekció esetén a szerző nem vehet részt a folyamatban felülvizsgálat-vezető vagy jegyzőkönyvvezető szerepkörben.

3.2.5 Felülvizsgálatok sikertényezői

Számos faktor van, amely meghatározza a felülvizsgálatok sikerességét, ezek például a következők lehetnek:

- Világosan meghatározott célokat és mérhető kilépési feltételeket kell megadni. A résztvevők értékelése sohasem lehet a felülvizsgálat célja
- A megfelelő felülvizsgálati típus kiválasztása, amely alkalmas a célok elérésére, és a munkatermékek típusához jól illeszkedik, továbbá megfelel a résztvevőknek és a projekt igényeinek és környezetének.
- A felülvizsgálatokat kisebb darabokon kell végezni, hogy a felülvizsgálók koncentrációja ne csökkenjen az egyéni felkészülés során és/vagy a felülvizsgálati értekezlet során (amennyiben erre sor kerül)
- Visszajelzést kell biztosítani a felülvizsgálók felől az érdekelt feleknek és a szerzőknek, hogy javítani tudják a terméket és a tevékenységeiket (lásd a 3.2.1. fejezetet)
- A résztvevők elegendő időt kapjanak a felülvizsgálatra történő felkészülésre
- A menedzsment támogassa a felülvizsgálat folyamatát
- A felülvizsgálatok elvégzése legyen része a szervezeti kultúrának, ezzel elősegítve a tanulást és a folyamatok javítását
- Megfelelő oktatást kell biztosítani az összes résztvevő számára, hogy tisztában legyenek a szerepük betöltésével
- Értekezletek lefolyásának támogatása

4. Tesztelemzés és műszaki teszttervezés 390 perc

Kulcsszavak

állapotátmenet-tesztelés, döntési tábla tesztelés, együttműködés-alapú tesztmegközelítés, ekvivalenciapartícionálás, elágazás lefedettség, elfogadási feltételek, elfogadásiteszt-vezérelt fejlesztés, ellenőrzőlista-alapú tesztelés, fehérdoboz-teszttechnika, feketedoboz-teszttechnika, felderítő tesztelés, határérték-elemzés, hibasejtés, lefedettség, lefedettségi elem, tapasztalatalapú teszttechnika, teszttechnika, utasításlefedettség

Tanulási célok a 4. fejezethez

4.1 Teszttechnikák áttekintése

AK-4.1.1 (K2) Tegyen különbséget a feketedoboz-, fehérdoboz- és a tapasztalatalapú teszttechnikák között

4.2 Feketedoboz-teszttechnikák

AK-4.2.1 (K3) Alkalmazza az ekvivalenciapartícionálást tesztesetek származtatására

AK-4.2.2 (K3) Alkalmazza a határérték-elemzést tesztesetek származtatására

AK-4.2.3 (K3) Alkalmazza a döntési tábla tesztelést tesztesetek származtatására

AK-4.2.4 (K3) Alkalmazza az állapotátmenet-tesztelést tesztesetek származtatására

4.3 Fehérdoboz-teszttechnikák

AK-4.3.1 (K2) Magyarázza el az utasítástesztelést

AK-4.3.2 (K2) Magyarázza el az elágazás tesztelést

AK-4.3.3 (K2) Magyarázza el a fehérdoboz tesztelés értékét

4.4 Tapasztalatalapú teszttechnikák

AK-4.4.1 (K2) Magyarázza el a hibasejtést

AK-4.4.2 (K2) Magyarázza el a felderítő tesztelést

AK-4.4.3 (K2) Magyarázza el az ellenőrzőlista-alapú tesztelést

4.5 Együttműködés-alapú tesztmegközelítések

AK-4.5.1 (K2) Magyarázza el a fejlesztőkkel és üzleti képviselőkkel együttműködésben történő fejlesztői történetek írásának mikéntjét

AK-4.5.2 (K2) Osztályozza az elfogadási feltételek írásának különböző lehetőségeit

AK-4.5.3 (K3) Alkalmazza az elfogadásiteszt-vezérelt fejlesztést (ATDD) tesztesetek származtatására

4.1 Teszttechnikák áttekintése

A teszttechnikák támogatják a tesztelőt a tesztelemzésben (mit kell tesztelni), illetve a műszaki teszttervezésben (hogyan kell tesztelni). A teszttechnikák segítenek a tesztesetek egy relatív kicsi, de elégséges halmazának szisztematikus kialakításában. A teszttechnikák továbbá a tesztelemzés és műszaki teszttervezés során segítik a tesztelőt a tesztfeltételek meghatározásában, a lefedettségi elemek és a tesztadatok azonosításában. További információ a teszttechnikákkal és a hozzájuk tartozó eszközökkel kapcsolatban az ISO/IEC/IEEE 29119-4-es szabványban található, illetve az alábbi referenciákban: Beizer 1990, Craig 2002, Copeland 2004, Koomen 2006, Jorgensen 2014, Ammann 2016, Forgács 2019.

Jelen tantervben a teszttechnikákat a következőképpen osztályozzuk: feketedoboz, fehérdoboz és tapasztalatalapú.

A **feketedoboz-teszttechnikák** (specifikációalapú technikákként is ismertek) alapvetően a teszt tárgyának adott viselkedését elemzik a belső struktúrájára történő hivatkozás nélkül. Ezért a tesztesetek függetlenek a szoftver implementálásának módjától. Ebből adódóan, ha az implementáció megváltozik, azonban az elvárt viselkedés ugyanaz marad, a tesztesetek továbbra is használhatók.

A **fehérdoboz-teszttechnikák** (struktúraalapú technikákként is ismertek) alapvetően a teszt tárgyának belső struktúráját és feldolgozását elemzik. Mivel a tesztesetek függenek a szoftver tervezésének módjától, kizárólag a teszt tárgyának megtervezése és megvalósítása után hozhatók létre.

A **tapasztalatalapú teszttechnikák** hatékonyan használják a tesztelők tudását és tapasztalatát a tesztesetek műszaki tervezéséhez és implementálásához. Ezen technikák hatékonysága nagyban függ a tesztelő képességeitől. A tapasztalatalapú teszttechnikák képesek észlelni olyan hibákat, amiket a feketedoboz- vagy a fehérdoboz-teszttechnikák figyelmen kívül hagynának. Ennek megfelelően a tapasztalatalapú teszttechnikák kiegészítik a feketedoboz- és fehérdoboz-teszttechnikákat.

4.2 Feketedoboz-teszttechnikák

A következő alfejezetekben tárgyalt, gyakran használt feketedoboz-teszttechnikák a következők:

- Ekvivalenciapartícionálás
- Határérték-elemzés
- Döntési tábla tesztelés
- Állapotátmenet-tesztelés

4.2.1 Ekvivalenciapartícionálás

Az ekvivalenciapartícionálás (EP) az adatokat partíciókra (úgynevezett ekvivalenciapartíciókra) bontja azon elvárás alapján, hogy a teszt tárgya egy adott partíció minden elemét ugyanolyan módon dolgozza fel. Ezen technika mögött az az elmélet áll, hogy ha egy teszteset, amely egy ekvivalenciapartíció egy értékét teszteli talál egy hibát, akkor ezt a hibát azoknak a teszteseteknek is meg kell találniuk, amik ugyanazon partíció bármely más értékét tesztelik. Ebből adódóan minden egyes partícióra elégséges egy teszt.

Ekvivalenciapartíciókat minden olyan adatelemhez meghatározhatunk, amely a teszt tárgyához kapcsolódik, beleértve a bemeneteket, kimeneteket, konfigurációs elemeket, belső értékeket, idővel kapcsolatos értékeket és interfész paramétereket. A partíciók lehetnek folyamatosak vagy diszkrétek, rendezettek vagy rendezetlenek, végesek vagy végtelenek. A partíciók között nem lehet átfedés és nem lehetnek üres halmazok.

Ha a teszt tárgya egyszerű, az EP is egyszerű lehet, de a gyakorlatban sokszor bonyolult megérteni, hogy a teszt tárgya hogyan fog kezelni különböző értékeket. Ezért megfelelő körültekintéssel kell elvégezni a partícionálást.

Az érvényes értékeket tartalmazó partíciót érvényes partíciónak nevezzük. Az érvénytelen értékeket tartalmazó partíciót érvénytelen partíciónak nevezzük. Az érvényes és érvénytelen értékek definíciói változhatnak csapatok és szervezetek között. Érvényes értékek lehetnek például azok, amelyeket a teszt tárgyának fel kell dolgoznia, vagy amelyekre a specifikáció meghatározza a feldolgozásukat. Érvénytelen értékek lehetnek azok, amelyeket a teszt tárgyának figyelmen kívül kell hagynia vagy el kell utasítania, vagy amelyekhez a teszt tárgyának specifikációjában nincs meghatározva feldolgozás.

Az EP-ben az ekvivalenciapartíciók a lefedettségi elemek. Ahhoz, hogy ezzel a technikával elérjük a 100%-os lefedettséget, a teszteseteknek az összes azonosított partíciót (az érvénytelen partíciókat is beleértve) le kell fedniük legalább egyszer. A lefedettséget a legalább egy értékkel tesztelt partíciók számának és az összes azonosított partíció számának hányadosaként határozzuk meg, százalékban kifejezve.

Sok esetben a teszt tárgya több partícióhalmazt tartalmaz (például több bemeneti paraméterrel rendelkezik), ami azt jelenti, hogy a teszteset különböző partícióhalmazokból fed le partíciókat. A legegyszerűbb lefedési kritérium több partícióhalmaz esetén az Each Choice lefedettség (Ammann 2016). Az Each Choice lefedettség megköveteli, hogy a tesztesetek minden egyes partícióhalmaz minden egyes partícióját legalább egyszer teszteljék. Az Each Choice lefedettség nem veszi figyelembe a partíciókombinációkat.

4.2.2 Határérték-elemzés

A határérték-elemzés (angol rövidítése BVA) egy olyan technika, amely az ekvivalenciapartíciók határainak tesztelésén alapul. Ezért csak rendezett partíciókra alkalmazható. A partíció minimum és maximum értékei a határértékek. Határérték-elemzés esetében, ha két elem ugyanazon partícióhoz tartozik, minden köztük lévő elemnek is ugyanehhez a partícióhoz kell tartoznia.

A határérték-elemzés azért fókuszál a partíciók határértékeire, mert a fejlesztők nagyobb eséllyel vétenek hibákat ezekkel a szélsőértékekkel. A határérték-elemzés által megtalált tipikus hibák ott találhatók, ahol az implementált határok a szándékolt pozíciójuk alá vagy fölé kerülnek, vagy teljesen kihagyják őket.

Eme tanterv a határérték-elemzés két verzióját fedi le: a 2-pontos és 3-pontos határérték-elemzést. Ezek a 100%-os lefedettség elérése érdekében tesztelendő határonkénti lefedett elemek tekintetében különböznek.

A 2-pontos határérték-elemzésben (Craig 2002, Myers 2011) minden határértékhez két lefedettségi elem tartozik: az adott határérték és a szomszédos partícióhoz tartozó legközelebbi szomszédja. Ahhoz, hogy a 2-pontos határérték-elemzéssel elérjük a 100%-os lefedettséget, a teszteseteknek minden lefedettségi elemet tesztelniük kell, tehát minden azonosított határértéket. A lefedettséget a tesztelt határértékek számának és az összes azonosított határérték számának hányadosaként határozzuk meg, százalékban kifejezve.

A 3-pontos határérték-elemzésben (Koomen 2006, O'Regan 2019) minden határértékhez három lefedettségi elem tartozik: az adott határérték és mindkét szomszédja. Ezért a 3-pontos határérték-elemzésben lehet, hogy néhány lefedettségi elem nem határérték. Ahhoz, hogy a 3-pontos határérték-elemzéssel elérjük a 100%-os lefedettséget, a teszteseteknek minden lefedettségi elemet tesztelniük kell, tehát minden azonosított határértéket és szomszédaikat. A lefedettséget a tesztelt határértékek illetve szomszédaik számának és az összes azonosított határérték illetve szomszédaik számának hányadosaként határozzuk meg, százalékban kifejezve.

A 3-pontos határérték-elemzés szigorúbb a 2-pontosnál, mivel megtalálhat olyan hibákat, amelyeket a 2-pontos határérték-elemzés nem vett észre. Például, ha az "if($x \leq 10$)..." utasítást helytelenül úgy implementálják, hogy "if ($x = 10$)...", a 2-pontos határérték-elemzéssel származtatott tesztadatok ($x = 10$, $x = 11$) nem tudják megtalálni a hibát. Azonban a 3-pontos határérték-elemzésből származtatott $x = 9$ valószínűleg képes rá.

4.2.3 Döntési tábla tesztelés

A döntési táblákat olyan rendszerkövetelmények implementációjának tesztelésére használják, amelyek meghatározzák, hogy a feltételek különböző kombinációi hogyan állítják elő a különböző kimeneteket. A döntési táblák hatékonyan alkalmazhatók komplex logika, pl. üzleti szabályok rögzítésére.

Döntési táblák készítésekor meghatározzák a rendszerben található feltételeket és az általuk eredményezett műveleteket. Ezek alkotják a táblázat sorait. Mindegyik oszlop egy döntési szabálynak felel meg, ami a feltételek egy egyedi kombinációját határozza meg, a hozzá rendelt műveletekkel együtt. A korlátozott bemenetű döntési táblákban minden feltétel és művelet (az irreleváns, illetve kivitelezhetetlen értékeken kívül; lásd alább) kétértékű (igaz vagy hamis). Alternatívaként, a bővített bemenetű döntési táblákban a feltételek, illetve műveletek többértékűek is lehetnek (pl. számtartományok, ekvivalenciapartíciók, diszkrét értékek).

A feltételek jelölése a következő: “I” (igaz) jelenti azt, hogy a feltétel teljesül. “H” (hamis) jelenti, hogy a feltétel nem teljesül. A “-” azt jelenti, hogy a feltétel értéke irreleváns a művelet kimenetele szempontjából. “N/A’ jelenti azt, hogy a feltétel kivitelezhetetlen egy adott szabályhoz. Műveletek esetében: “X” jelenti, hogy a műveletet végre kell hajtani, az üresen hagyott cella pedig azt jelenti, hogy a műveletet nem kell végrehajtani. Ezeken kívül más jelölések is használhatók.

Egy teljes döntési táblának elég oszlopa van ahhoz, hogy a feltételek minden kombinációját lefedje. A tábla leegyszerűsíthető, ha a feltételek kivitelezhetetlen kombinációit tartalmazó oszlopokat töröljük. A tábla minimalizálható, ha egy oszlopba fésüljük az olyan oszlopokat, amelyekben néhány feltétel nem befolyásolja a kimenetet. A döntési táblák minimalizálási algoritmusai túlmutatnak ezen tanterv anyagán.

Döntési tábla tesztelésben a lefedettségi elemek a feltételek kivitelezhető kombinációit tartalmazó oszlopok. A 100%-os lefedettség elérése érdekében a teszteseteknek minden ilyen oszlopot tesztelniük kell. A lefedettséget a tesztelt oszlopok számának és az összes kivitelezhető oszlop számának hányadosaként határozzuk meg, százalékban kifejezve.

A döntési tábla tesztelés erőssége, hogy módszeres megközelítést biztosít az összes feltételkombináció azonosítására, amelyek közül néhányat egyébként figyelmen kívül hagyhatnánk. Ezen kívül segít megtalálni a követelményekben lévő réseket vagy ellentmondásokat. A szabályok száma a feltételek számának növekedésével exponenciálisan nő, ezért ha sok feltétel van, az összes döntési szabály végrehajtása időigényes lehet. Ilyen esetben, a végrehajtandó szabályok csökkentése érdekében, minimalizált döntési tábla vagy kockázatalapú megközelítés használható.

4.2.4 Állapotátmenet-tesztelés

Az állapotátmenet-diagram egy rendszer viselkedését modellezi a lehetséges állapotok és az érvényes állapotátmenetek ábrázolásával. Az átmenetet egy esemény váltja ki, amelyet egy őrfeltétel is kiegészíthet. Az átmenetekről feltételezzük, hogy azonnaliak, és néha szoftverműveletet eredményezhetnek. Az átmenet jelölésének megszokott szintaxisa a következő: “esemény [őrfeltétel] / művelet”. Amennyiben az őrfeltételek, illetve a műveletek nem léteznek vagy irrelevánsak a tesztelő szempontjából, úgy elhagyhatók.

Az állapottábla az állapotátmenet-diagrammal egyenértékű modell. Sorai az állapotokat reprezentálják, az oszlopai pedig az eseményeket (az őrfeltételekkel együtt, ha léteznek). A táblabejegyzések (cellák) az átmeneteket reprezentálják, és a célállapotokat tartalmazzák, illetve az eredményezett műveleteket, ha definiáltuk őket. Az állapotátmenet-diagrammal ellentétben, az állapottábla egyértelműen megjeleníti az érvénytelen átmeneteket is, amelyeket üres cella jelöl.

Az állapotátmenet-diagramon vagy állapottáblán alapuló teszteket általában események sorozataként határozzák meg, amely állapotváltozások (és szükség esetén műveletek) sorozatát eredményezi. Egy teszteset számos állapotátmenetet lefedhet, és többnyire le is fed.

Az állapotátmenet-teszteléshez sok lefedettségi feltétel létezik, amelyek közül ez a tanterv hármat tárgyal.

Összes állapot lefedettség esetén az állapotok a lefedettségi elemek. A 100%-os összes állapotlefedettség elérése érdekében a teszteseteknek biztosítaniuk kell, hogy minden állapotot érintsünk. A lefedettséget az érintett állapotok számának és az összes állapot számának hányadosaként határozzuk meg, százalékban kifejezve.

Érvényes átmenet lefedettség (0-lépéses lefedettségnek is hívják) esetén a lefedettségi elemek az egyedülálló érvényes átmenetek. A 100%-os érvényes átmenetlefedettség elérése érdekében a teszteseteknek minden érvényes átmenetet tesztelniük kell. A lefedettséget a felhasznált érvényes átmenetek és az összes érvényes átmenet számának hányadosaként határozzuk meg.

Összes átmenet lefedettség esetén a lefedettségi elemek az állapottáblában szereplő átmenetek összessége. A 100%-os átmenetlefedettség elérése érdekében a teszteseteknek minden érvényes átmenetet tesztelniük kell és meg kell kísérelniük végrehajtani az érvénytelen átmeneteket. Ha egy teszteset csak egyetlen érvénytelen átmenetet tesztel, akkor ez segíthet elkerülni a hibaelfedést, vagyis azt a helyzetet, amikor egy hiba megakadályozza egy másik hiba észrevételét. A lefedettséget a tesztesetek által felhasznált, vagy a végrehajtott tesztesetek által lefedni kívánt érvényes és érvénytelen átmenetek számának és az összes érvényes és érvénytelen átmenet számának hányadosaként határozzuk meg, százalékban kifejezve.

Az “összes állapotlefedettség” gyengébb, mint az “érvényes átmenetlefedettség”, mivel az előbbi tipikusan elérhető az összes átmenet tesztelése nélkül. Az “érvényes átmenetlefedettség” a legszélesebb körben alkalmazott lefedettségi feltétel. A teljes “összes átmenetlefedettség” elérése garantálja a teljes “összes állapotlefedettséget”. A teljes “összes átmenetlefedettség” garantálja mind a teljes “összes állapotlefedettséget”, mind a teljes “érvényes átmenetlefedettséget” és minimumkövetelménynek kell lennie a küldetés-, illetve a biztonságkritikus szoftverek esetében.

4.3 Fehérdoboz-teszttechnikák

A népszerűségük és egyszerűségük miatt ez az alfejezet két, kódhoz kapcsolódó fehérdoboz-teszttechnikára fókuszál:

- Utasítástesztelés
- Elágazási tesztelés

Léteznek szigorúbb technikák is, amelyeket néhány biztonságkritikus, küldetéskritikus vagy magas integritású környezetben alkalmaznak az alaposabb kódlefedettség elérése érdekében. Ezenkívül vannak olyan fehérdoboz-teszttechnikák, amelyeket magasabb tesztszinteken alkalmaznak (pl. API tesztelés), vagy amik nem kódhoz kapcsolódó lefedettséget használnak (pl. neurális hálózatok tesztelésénél a neuron lefedettség). Ezen technikákat nem tárgyaljuk ebben a tantervben.

4.3.1 Utasítástesztelés és utasításlefedettség

Utasítástesztelésben a lefedettségi elemek a végrehajtható utasítások. A cél olyan tesztesetek tervezése, amelyek addig futtatják a kódban lévő utasításokat, amíg el nem érnek egy elfogadható lefedettségi szintet. A lefedettséget a tesztesetek által felhasznált utasítások és a kódban lévő összes végrehajtható utasítás számának hányadosaként határozzuk meg, százalékban kifejezve.

Ha az utasításlefedettség elérte a 100%-ot, akkor ez biztosítja, hogy minden végrehajtható utasítás legalább egyszer felhasználásra került. Ez azt jelenti, hogy minden hibás utasítás végre lesz hajtva, ami meghibásodást okozhat, a hiba jelenlétét mutatva. Azonban egy utasítás végrehajtása nem minden esetben talál hibát. Például lehet, hogy nem találja meg az adatfüggő hibákat (pl. nullával való osztás, ami csak akkor bukik el, ha nulla a nevező). Ráadásul a 100%-os utasításlefedettség nem biztosítja az összes döntési logika tesztelését, például előfordulhat, hogy a tesztek nem futtatják az összes elágazást (lásd a 4.3.2. fejezetet).

4.3.2 Elágazási tesztelés és elágazás lefedettség

Az elágazás a vezérlés átadása a vezérlésifolyam-gráf két csomópontja között, amely megmutatja, hogy a teszt tárgyában milyen lehetséges sorozatokban futnak le a forráskód utasításai. Minden vezérlésátadás lehet feltétel nélküli (pl. szekvenciális kód) vagy feltételes (pl. döntési kimenet).

Az elágazási tesztelésben a lefedettségi elemek az elágazások. A cél olyan tesztesetek tervezése, amelyek addig futtatják a kódban lévő elágazásokat, amíg el nem érnek egy elfogadható lefedettségi szintet. A lefedettséget a tesztesetek által felhasznált elágazások és az összes elágazás számának hányadosaként határozzuk meg, százalékban kifejezve.

Ha az elágazáslefedettség elérte a 100%-ot, a tesztesetek a kódban lévő minden feltételes és feltétel nélküli elágazást futtattak. A feltételes elágazások tipikusan egy “ha…akkor” döntés igaz vagy hamis kimenetének, egy switch/case utasítás kimenetének vagy egy ciklus folytatására vonatkozó döntésnek felelnek meg. Azonban egy elágazás futtatása nem minden esetben találja meg a hibákat. Például lehetséges, hogy nem talál meg egy olyan hibát, amely a kód egy meghatározott útvonalának végrehajtása során lép fel.

Az elágazáslefedettség magában foglalja az utasításlefedettséget. Ez azt jelenti, hogy bármely olyan tesztesethalmaz, amely 100%-os elágazáslefedettséget ér el, eléri a 100%-os utasításlefedettséget is (de fordítva nem igaz).

4.3.3 A fehérdoboz tesztelés értéke

Az alapvető erősség, ami minden fehérdoboz-technikára vonatkozik az, hogy teszteléskor a teljes szoftverimplementációt figyelembe veszik, ami lehetővé teszi a hibák megtalálását még abban az esetben is, ha a szoftver specifikációja homályos, elavult vagy nem teljes. Ennek megfelelő gyengesége az, hogy ha a szoftver nem implementál egy vagy több követelményt, lehetséges, hogy a fehérdoboz tesztelés nem találja meg a mulasztás okozta hibákat (Watson 1996).

A fehérdoboz-technikák alkalmazhatók statikus tesztelésben (pl. a kód száraz tesztelése során). Megfelelnek olyan kódok felülvizsgálatára, amelyek még nem állnak készen a végrehajtásra (Hetzel 1988), illetve pszeudokódok és más, vezérlésifolyam-gráffal modellezhető magas szintű vagy felülről lefelé irányuló logika felülvizsgálatára.

Kizárólag a feketedoboz tesztelés végrehajtása nem biztosítja a tényleges kódlefedettség mérhetőségét. A fehérdoboz-lefedettségi eszközök lehetőséget adnak a lefedettség objektív mérésére, és megadják a szükséges információt az olyan további tesztek generálásához, amelyek növelik a lefedettséget és később növelhetik a kódba vetett bizalmat.

4.4 Tapasztalatalapú teszttechnikák

A következő alfejezetekben tárgyalt, gyakran használt tapasztalatalapú teszttechnikák az alábbiak:

- Hibasejtés
- Felderítő tesztelés
- Ellenőrzőlista-alapú tesztelés

4.4.1 Hibasejtés

A hibasejtés egy technika, mely segítségével a tesztelő tudása alapján előre jelzik az emberi eredetű hibák (error), a hibák (defect) és a meghibásodások (failure) előfordulását, beleértve:

- Hogyan működött az alkalmazás a múltban
- Emberi eredetű hibák típusai, amelyeket a fejlesztők hajlamosak elkövetni, és az ezekből adódó hibák típusai
- Más, hasonló alkalmazásokban előforduló meghibásodások típusai

Általánosságban, az emberi eredetű hibák, hibák és meghibásodások összefüggésben lehetnek: a bemenettel (pl. nem elfogadott helyes bemenet, rossz vagy hiányzó paraméterek), a kimenettel (pl. rossz formátum, rossz eredmény), a logikával (pl. hiányzó esetek, rossz operátor), a számítással (pl. helytelen operandus, rossz számítás), az interfészekkel (pl. nem egyeznek a paraméterek, inkompatibilis típusok), vagy az adatokkal (pl. helytelen inicializáció, rossz típus).

A hibasejtés megvalósításának egy módszeres megközelítése a hibatámadás. Ez a technika megköveteli, hogy a tesztelő készítsen vagy szerezzen egy listát a lehetséges emberi eredetű hibákról, hibákról, illetve meghibásodásokról, és olyan teszteket tervezzen, amelyek azonosítják az emberi eredetű hibák által okozott hibákat, felfedik a hibákat vagy előidézik a meghibásodásokat. Ezen listák készülhetnek tapasztalat alapján, hibákról és meghibásodásokról rendelkezésre álló adatok alapján, valamint a szoftver meghibásodásával kapcsolatos általános ismeretanyag alapján.

A hibasejtéssel és hibatámadással kapcsolatos további információkért lásd: Whittaker 2002, Whittaker 2003, Andrews 2006.

4.4.2 Felderítő tesztelés

Felderítő tesztelés során a tesztek tervezése, végrehajtása és kiértékelése egy időben történik, miközben a tesztelő ismereteket gyűjt a teszt tárgyáról. A tesztelést arra használják, hogy további információkat szerezzenek a teszt tárgyáról, célzott tesztekkel mélyebben megismerjék azt, illetve, hogy a még nem tesztelt területekhez készítsenek teszteket.

A tesztelés strukturálása érdekében a felderítő tesztelést néha munkamenet-alapú teszteléssel hajtják végre. A munkamenet-alapú megközelítéssel a felderítő tesztelést egy meghatározott időkereten belül végzik. A tesztelő egy tesztcélokat tartalmazó tesztvázlatot használ, ami irányvonalat ad a teszteléshez. A tesztszakaszt általában egy összefoglaló eligazítás követi, ami magában foglalja a tesztelő és a teszteredményben érdekelt felek közötti megbeszélést. Ebben a megközelítésben a tesztcélokat magas szintű tesztfeltételekként lehet kezelni. A lefedettségi elemeket a tesztszakasz során azonosítják és hajtják végre. A követett lépések és a felfedezések dokumentálására tesztszakasz-jegyzőkönyv használható.

A felderítő tesztelés hasznos, amikor kevés vagy nem megfelelő specifikáció áll csak rendelkezésre, vagy jelentős az időnyomás a tesztelésen. A felderítő tesztelés más, formálisabb teszttechnikák kiegészítéseként is hasznos. A felderítő tesztelés eredményesebb lesz, ha a tesztelő tapasztalt, rendelkezik szakterületi tudással, illetve magas szintű alapvető képességekkel, mint az analitikus képesség, kíváncsiság és kreativitás (lásd az 1.5.1. fejezetet).

A felderítő tesztelés magában foglalhatja más teszttechnikák (pl. ekvivalenciapartícionálás) használatát. A felderítő teszteléssel kapcsolatos további információkért lásd: Kaner 1999, Whittaker 2009, Hendrickson 2013.

4.4.3 Ellenőrzőlista-alapú tesztelés

Ellenőrzőlista-alapú tesztelés során a tesztelő egy ellenőrzőlistában található tesztfeltételek lefedésére tervezi, valósítja meg és hajtja végre a teszteket. Az ellenőrzőlisták készíthetők tapasztalat alapján, a felhasználó számára fontos szempontokról meglévő tudás alapján vagy a szoftver meghibásodásának okairól és azok jellegéről alkotott kép alapján. Az ellenőrzőlisták nem tartalmazhatnak olyan elemeket, amelyek automatikusan ellenőrizhetők, vagy inkább belépési vagy kilépési feltételek lehetnének, esetleg amelyek túl általánosak (Brykczynski 1999).

Az ellenőrzőlista elemeit gyakran kérdések formájában fogalmazzák meg. Lehetővé kell tenni, hogy minden egyes elem elkülönülten és közvetlenül ellenőrizhető legyen. Ezen elemek utalhatnak követelményekre, grafikus interfész tulajdonságokra, minőségjellemzőkre vagy a tesztfeltételek más formáira. Az ellenőrzőlisták különböző teszttípusok támogatására készíthetők, beleértve a funkcionális, illetve a nemfunkcionális tesztelést is (pl. a használhatósági tesztelés 10 heurisztikája (Nielsen 1994)).

Néhány ellenőrzőlista bejegyzés idővel fokozatosan veszít az eredményességéből, mivel a fejlesztők megtanulják elkerülni, hogy ugyanazokat az emberi eredetű hibákat kövessék el. Új bejegyzések hozzáadására is szükség lehet, hogy tükrözze az új, nagy súlyosságú hibákat. Ezért az ellenőrzőlistákat rendszeresen frissíteni kell a hibaelemzés alapján. Viszont arra figyelni kell, hogy az ellenőrzőlisták ne váljanak túl hosszúvá (Gawande 2009).

Részletes tesztesetek hiányában, az ellenőrzőlista-alapú tesztelés irányvonalat és bizonyos fokú következetességet adhat a tesztelésnek. Ha az ellenőrzőlisták magas szintűek, akkor a tényleges tesztelésben előfordulhatnak eltérések, amely potenciálisan nagyobb lefedettséget, de kisebb mértékű ismételhetőséget eredményez.

4.5 Együttműködés-alapú tesztmegközelítések

Minden fent említett technikának (lásd a 4.2., 4.3. és 4.4. fejezeteket) van egy sajátos célja a hibafelismerés tekintetében. Az együttműködés-alapú megközelítések viszont a hibamegelőzésre is hangsúlyt fektetnek az együttműködés és kommunikáció révén.

4.5.1 Együttműködésen alapuló felhasználói történet-írás

A felhasználói történet egy olyan funkciót reprezentál, ami a rendszer vagy szoftver felhasználójának vagy vásárlójának lesz értékes. A felhasználói történeteknek három kritikus szempontja van (Jeffries 2000), amiket a “3 C-nek” neveznek:

- Kártya (Card) - ez írja le a felhasználói történetet (pl. egy fizikai kártya vagy bejegyzés egy elektronikus táblán)
- Beszélgetés (Conversation) - elmagyarázza a szoftver használatát (lehet szóbeli vagy dokumentált)
- Megerősítés (Confirmation) - az elfogadási feltételek (lásd a 4.5.2. fejezetet)

A felhasználói történet leggyakoribb formája: “Mint [szerepkör], szeretném [elérendő cél], azért, hogy [a szerepkörnek szükséges üzleti érték]”, melyet az elfogadási feltételek követnek.

A felhasználói történetek közös megírása során alkalmazhatók olyan technikák, mint az ötletelés (brainstorming) vagy gondolattérképek készítése. Az együttműködés lehetővé teszi, hogy a csapat egy közös elképzeléssel rendelkezzen a szállítandó termékről az alábbi három szempont figyelembevételével: üzlet, fejlesztés és tesztelés.

A jó felhasználói történetek az alábbi jellemzőkkel írhatók le (INVEST): Független (Independent), Tárgyalható (Negotiable), Értékkel bír (Valuable), Becsülhető (Estimable), Kicsi (Small), Tesztelhető (Testable) . Ha egy érdekelt fél nem tudja, hogyan kell egy felhasználói történetet tesztelni, azt jelezheti, hogy a felhasználói történet nem elég érthető, vagy, hogy számára nem bír értékkel, vagy pedig, hogy az érdekelt félnek csak segítségre van szüksége a teszteléshez (Wake 2003).

4.5.2 Elfogadási feltételek

Egy felhasználói történet elfogadási feltételei azok a feltételek, amelyeknek a felhasználói történet megvalósítása során teljesülniük kell ahhoz, hogy az érdekelt felek elfogadják. Így az elfogadási feltételeket tekinthetjük tesztfeltételeknek, amelyeket a teszteknek ki kell elégíteniük. Az elfogadási feltételek általában a Beszélgetés eredményei (lásd a 4.5.1. fejezetet).

Az elfogadási feltételek:

- Segítenek a felhasználói történet hatókörének meghatározásában
- Segítenek az érdekelt felek közötti megegyezés elérésében
- Segítenek a pozitív és negatív forgatókönyvek leírásában
- A felhasználói történet elfogadási tesztelés alapjául szolgálnak (lásd a 4.5.3. fejezetet)
- Lehetővé teszik a pontos tervezést és becslést

Többféle mód létezik az elfogadási feltételek megírására. A két leggyakoribb forma a következő:

- Forgatókönyv-orientált (pl. a BDD-ben használt Given/When/Then formátum, lásd a 2.1.3. fejezetet)
- Szabály-orientált (pl. verifikációs lista vagy bemenet-kimenet leképezés táblázatos formában)

A legtöbb elfogadási feltétel leírható a két formátum egyikének alkalmazásával. Mindazonáltal, a csapat használhat más, saját formátumot mindaddig, amíg az elfogadási feltétel jól definiált és egyértelmű marad.

4.5.3 Elfogadásiteszt-vezérelt fejlesztés (ATDD)

Az ATDD egy test-first megközelítés (lásd a 2.1.3. fejezetet). A teszteseteket a felhasználói történetek implementálása előtt készítik el. A teszteseteket különböző nézőponttal rendelkező csapattagok írják, például ügyfelek, fejlesztők és tesztelők (Adzic 2009). A tesztesetek lehetnek manuálisak vagy automatizáltak.

Az első lépés a specifikációs workshop, ahol a csapattagok elemzik, megbeszélik és megírják a felhasználói történetet és az elfogadási feltételeket (amennyiben ezek még nem definiáltak). A felhasználói történet hiányosságai, kétértelműségei vagy hibái eme folyamat során kerülnek feloldásra. A következő lépés a tesztesetek elkészítése. Ez történhet az egész csapat együttes munkájával, vagy egyénileg, a tesztelő által. A tesztesetek az elfogadási feltételeken alapulnak és példaként szolgálnak arra, hogyan működik a szoftver. Ez segíti a csapatot a fejlesztői történet helyes megvalósításában.

Mivel a példák és a tesztek ugyanazok, ezeket a fogalmakat gyakran egymás szinonimájaként használják. A műszaki teszttervezés során a 4.2-es, 4.3-as, illetve 4.4-es fejezetben leírt teszttechnikák használhatók.

Általában az első tesztesetek pozitívak, amelyek megerősítik a helyes viselkedést kivételek és hibafeltételek nélkül, és tartalmazzák a tevékenységek azon sorozatát, amelyeket végrehajtanak, ha minden az elvárt módon történik. Miután a pozitív tesztek készen vannak, a csapatnak negatív tesztelést kell végrehajtania. Végül pedig le kell fedni a nemfunkcionális minőségjellemzőket is (például teljesítményhatékonyság, használhatóság). A teszteseteket olyan módon kell megírni, hogy az érdekelt felek számára érthető legyen. Általában a tesztesetek természetes nyelven íródott mondatokat tartalmaznak, illetve magukban foglalják a szükséges előfeltételeket (ha vannak), a bemeneteket, illetve az utófeltételeket.

A teszteseteknek a felhasználói történet összes jellemzőjét le kell fedniük, viszont annál többet nem. Azonban az elfogadási feltételek részletezhetnek néhány, a felhasználói történetben leírt problémát. Továbbá, két teszteset sem írhatja le a felhasználói történet ugyanazon jellemzőjét.

Ha a tesztesetek a tesztautomatizálási keretrendszer által támogatott formátumban készültek, a fejlesztők képesek automatizálni őket, miközben a felhasználói történetben leírt funkciót implementálják. Az elfogadási tesztek ezzel végrehajtható követelményekké válnak.

5. Teszttevékenységek menedzselése 335 perc

Kulcsszavak

belépési feltételek, hibajelentés, hibamenedzsment, kilépési feltételek, kockázatalapú tesztelés, kockázat, kockázati szint, kockázatazonosítás, kockázatelemzés, kockázatértékelés, kockázatfelügyelet, kockázatirányítás, kockázatmenedzsment, kockázatmérséklés, összefoglaló tesztjelentés, projektkockázat, termékkockázat, tesztelési kvadránsok, tesztelőrehaladási jelentés, tesztfelügyelet, tesztirányítás, tesztmegközelítés, tesztpiramis, teszterv, teszttervezés

Tanulási célok az 5. fejezethez

5.1. Teszttervezés

AK-5.1.1 (K2) Szemléltesse a teszterv célját és tartalmát

AK-5.1.2 (K1) Ismerje fel, hogyan ad hozzá értéket a tesztelő az iteráció- és kiadástervezéshez

AK-5.1.3 (K2) Hasonlítsa össze a belépési és kilépési feltételeket

AK-5.1.4 (K3) Használjon becslési technikákat a szükséges tesztráfordítás kiszámításához

AK-5.1.5 (K3) Alkalmazzon teszteset-priorizálást

AK-5.1.6 (K1) Idézze fel a tesztpiramis fogalmait

AK-5.1.7 (K2) Foglalja össze a tesztelési kvadránsokat és azok kapcsolatát a tesztszintekkel és teszttípusokkal

5.2. Kockázatmenedzsment

AK-5.2.1 (K1) Azonosítsa a kockázati szintet a kockázat valószínűségének és hatásának felhasználásával

AK-5.2.2 (K2) Különböztesse meg a projektkockázatokat és termékkockázatokat

AK-5.2.3 (K2) Magyarázza el, hogyan befolyásolhatja a termékkockázat-elemzés a tesztelés alaposságát és hatókörét

AK-5.2.4 (K2) Magyarázza el, milyen intézkedéseket lehet tenni az elemzett termékkockázatokra reagálva

5.3. Tesztfelügyelet, tesztirányítás és tesztlezárás

AK-5.3.1 (K1) Idézze fel a teszteléshez használt metrikákat

AK-5.3.2 (K2) Foglalja össze a tesztjelentések célját, tartalmát és célközönségét

AK-5.3.3 (K2) Szemléltesse, hogyan lehet kommunikálni a tesztelés státuszát

5.4. Konfigurációmenedzsment

AK-5.4.1 (K2) Foglalja össze, hogyan támogatja a konfigurációmenedzsment a tesztelést

5.5. Hibamenedzsment

AK-5.5.1 (K3) Készítsen egy hibajelentést

5.1 Teszttervezés

5.1.1 Teszterv célja és tartalma

Egy teszterv leírja a tesztprojekt céljait, erőforrásait és folyamatait. Egy teszterv:

- Dokumentálja az eszközöket és az ütemtervet a tesztcélok eléréséhez
- Segít biztosítani, hogy a végrehajtott teszttevékenységek megfeleljenek az előírt feltételeknek
- Kommunikációs eszközként szolgál a csapattagok és más érdekelt felek számára
- Bemutatja, hogy a tesztelés megfelel a meglévő tesztelési irányelveknek és tesztstratégiának (vagy megmagyarázza, hogy miért tér el a tesztelés ezektől).

A teszttervezés irányítja a tesztelők gondolkodását, és szembesíti őket a jövőbeli kihívásokkal, amelyek kapcsolódnak a kockázatokhoz, ütemtervekhez, emberekhez, eszközökhöz, költségekhez, ráfordításokhoz stb. A teszterv elkészítésének folyamata hasznos módja annak, hogy átgondoljuk a ráfordításokat, amelyekre szükség van a tesztprojekt céljainak eléréséhez.

A teszterv tipikusan a következőket tartalmazza:

- A tesztelés kontextusa (pl. hatókör, tesztcélok, korlátok, tesztbázis)
- Feltételezések és korlátok a tesztprojektben
- Érdekelt fél (pl. szerepek, felelősségek, relevancia a teszteléshez, felvételi és képzési igények)
- Kommunikáció (pl. kommunikáció formái és gyakorisága, dokumentációs sablonok)
- Kockázati nyilvántartás (pl. termékkockázatok, projektkockázatok)
- Tesztmegközelítés (pl. tesztszintek, teszttípusok, teszttechnikák, teszttermékek, belépési feltételek és kilépési feltételek, független tesztelés, gyűjtendő metrikák, tesztadat követelmények, tesztkörnyezet követelmények, eltérések a szervezeti tesztelési irányelvektől és tesztstratégiától)
- Költségvetés és ütemterv

További részletek a teszttervről és annak tartalmáról megtalálhatók az ISO/IEC/IEEE 29119-3 szabványban.

5.1.2 A tesztelő hozzájárulása az iteráció- és kiadástervezéshez

Az iteratív szoftverfejlesztési életciklusokban általában két fajta tervezés történik: kiadástervezés és iterációtervezés.

A kiadástervezés előretekint a termék kiadására, meghatározza és újra meghatározza a termék-teendőlistát, és magában foglalhatja a nagyobb felhasználói történetek felbontását egy sor kisebb felhasználói történetre. Ez szolgáltatja továbbá az alapját a tesztmegközelítésnek és teszttervnek az összes iteráción keresztül. A kiadástervezésben részt vevő tesztelők részt vesznek a tesztelhető felhasználói történetek és elfogadási feltételek írásában (lásd a 4.5. fejezetet), részt vesznek a projekt- és minőségkockázat-elemzésekben (lásd 5.2. fejezet), megbecsülik a tesztráfordítást a felhasználói történetekkel kapcsolatban (lásd az 5.1.4. fejezetet), meghatározzák a tesztmegközelítést, és megtervezik a tesztelést a kiadáshoz.

Az iterációtervezés egyetlen iteráció végére tekint előre, és az iteráció-teendőlistával foglalkozik. Az iterációtervezésben részt vevő tesztelők részt vesznek a felhasználói történetek részletes kockázatelemzésében, meghatározzák a felhasználói történetek tesztelhetőségét, felbontják a felhasználói történeteket feladatokra (különösen tesztelési feladatokra), megbecsülik a tesztráfordítást minden tesztelési feladatra, és azonosítják és finomítják a teszt tárgyának funkcionális és nemfunkcionális aspektusait.

5.1.3 Belépési és kilépési feltételek

A belépési feltételek meghatározzák az adott tevékenység elvégzésének előfeltételeit. Ha a belépési feltételek nem teljesülnek, valószínű, hogy a tevékenység nehezebb, időigényesebb, költségesebb és kockázatosabb lesz. A kilépési feltételek meghatározzák, hogy mit kell elérni ahhoz, hogy egy tevékenységet befejezettnek nyilvánítsunk. A belépési és kilépési feltételeket minden tesztszintre meg kell határozni, és eltérnek a tesztcéloktól függően.

A tipikus belépési feltételek közé tartozik: az erőforrások rendelkezésre állása (pl. emberek, eszközök, környezetek, tesztadatok, költségvetés, idő), a tesztelhetőség rendelkezésre állása (pl. tesztbázis, tesztelhető követelmények, felhasználói történetek, tesztesetek), és a teszt tárgyának kezdeti minőségi szintje (pl. minden smoke teszt sikeres volt).

A tipikus kilépési feltételek közé tartozik: az alaposság mérőszámai (pl. elérhető lefedettségi szint, megoldatlan hibák száma, hibasűrűség, bukott tesztesetek száma), és a teljesítési feltételek (pl. a tervezett teszteket végrehajtották, statikus tesztelést végeztek, minden talált hibát jelentettek, minden regressziós teszt automatizált).

Az idő vagy a költségvetés elfogyása is érvényes kilépési feltételnek tekinthető. Még akkor is elfogadható a tesztelés teljesítése ilyen körülmények között, ha az érdekelt felek áttekintették és elfogadták annak kockázatát, hogy további tesztelés nélkül engedik ki a terméket.

Agilis szoftverfejlesztésben a kilépési feltételeket gyakran Definition of Done-nak nevezik, amely meghatározza a csapat objektív metrikáit egy kiadható elemhez. A belépési feltételeket, amelyeket egy felhasználói történetnek teljesítenie kell ahhoz, hogy elkezdhesse a fejlesztési és/vagy tesztelési tevékenységeket, Definition of Ready-nek nevezik.

5.1.4 Becslési technikák

A tesztráfordítás becslése magában foglalja a tesztprojekt céljainak eléréséhez szükséges teszttel kapcsolatos munka mennyiségének előrejelzését. Fontos, hogy világossá tegyük az érdekelt felek számára, hogy a becslés számos feltételezésen alapul, és mindig becslési hibával rendelkezik. A kisebb feladatokra történő becslés általában pontosabb, mint a nagyobbakéra. Ezért, amikor egy nagy feladatra becslést készítünk, azt felbontjuk egy sor kisebb feladatra, amelyekre aztán sorban becslést készíthetünk.

Ebben a tantervben a következő négy becslési technikát írjuk le.

Becslés arányok alapján. Ebben a metrikán alapuló technikában adatokat gyűjtenek a szervezet korábbi projektjeiről, ami lehetővé teszi, hogy „standard” arányokat vezessenek le hasonló projektekhez. A szervezet saját projektjeinek arányai (pl. történelmi adatokból vett) általában a legjobb forrást jelentik a becslési folyamatban. Ezeket a standard arányokat aztán felhasználhatjuk a tesztráfordítás becslésére az új projektben. Például, ha a korábbi projektben a fejlesztés-tesztráfordítás aránya 3:2 volt, és az aktuális projektben a fejlesztés-ráfordítás várhatóan 600 embernap lesz, akkor a tesztráfordítás 400 embernapnak becsülhető.

Extrapoláció. Ebben a metrikán alapuló technikában minél hamarabb végeznek méréseket az aktuális projektben az adatok gyűjtése érdekében. Elegendő megfigyeléssel rendelkezve a hátralévő munka szükséges ráfordítása közelíthető az adatok extrapolálásával (általában egy matematikai modell alkalmazásával). Ez a módszer nagyon alkalmas iteratív szoftverfejlesztési életciklusokban. Például, a csapat extrapolálhatja a tesztráfordítást a következő iterációban az utolsó három iteráció átlagos ráfordítása alapján.

Széleskörű Delphi. Ebben az iteratív, szakértőalapú technikában tapasztalatalapú becsléseket készítenek a szakértők. Minden szakértő elkülönülve becsüli meg a ráfordítást. Az eredményeket összegyűjtik, és ha olyan eltérések vannak, amelyek meghaladják az elfogadott határokat, akkor a szakértők megbeszélik

jelenlegi becsléseiket. Ezután minden szakértőt arra kérnek, hogy készítsenek egy új becslést ezen visszajelzések alapján, ismét elkülönülve. Ez a folyamat addig ismétlődik, amíg konszenzus nem alakul ki. A tervezési póker egy változata a széleskörű Delphi-nek, amelyet gyakran használnak agilis szoftverfejlesztésben. A tervezési pókerben a becsléseket általában olyan kártyák használatával készítik el, amelyek számokkal reprezentálják a ráfordítás nagyságát.

Hárompontos becslés. Ebben a szakértőalapú technikában három becslést készítenek a szakértők: a legoptimistább becslést (a), a legvalószínűbb becslést (m) és a legpesszimistább becslést (b). A végső becslés (E) az átlagos súlyozott számtani átlaguk. Ennek a technikának a legnépszerűbb változatában a becslés a következőképpen számítható ki: E = (a + 4*m + b) / 6. Ennek a technikának az az előnye, hogy lehetővé teszi a szakértők számára, hogy kiszámítsák a mérési hibát: SD = (b – a) / 6. Például, ha a becslések (emberórában) a következők: a=6, m=9 és b=18, akkor a végső becslés 10±2 emberóra (azaz 8 és 12 emberóra között), mert E = (6 + 4*9 + 18) / 6 = 10 és SD = (18 – 6) / 6 = 2.

Lásd (Kan 2003, Koomen 2006, Westfall 2009) ezeket és sok további tesztbecslési technikát.

5.1.5 Teszteset-priorizálás

Miután a teszteseteket és teszteljárásokat meghatározták és tesztkészletekbe rendezték, ezeket a tesztkészleteket egy tesztvégrehajtási ütemtervbe lehet szervezni, amely meghatározza, milyen sorrendben kell őket futtatni. A tesztesetek priorizálásakor különböző tényezőket lehet figyelembe venni. A leggyakrabban használt teszteset-priorizálási stratégiák a következők:

- Kockázatalapú priorizálás, ahol a tesztvégrehajtás sorrendje a kockázatelemzés eredményein alapul (lásd az 5.2.3. fejezetet). Elsőként azokat a teszteseteket hajtják végre, amelyek a legfontosabb kockázatokat fedik le.
- Lefedettségalapú priorizálás, ahol a tesztvégrehajtás sorrendje a lefedettségen alapul (pl. utasításlefedettség). Azokat a teszteseteket hajtják végre először, amelyek a legmagasabb lefedettséget érik el. Egy másik változatban, az úgynevezett *kiegészítő lefedettség-priorizálás*ban, először a legnagyobb lefedettséget elérő teszteset kerül végrehajtásra; minden további teszteset a lefedettséget legnagyobb mértékben kiegészítő teszteset.
- Követelményalapú priorizálás, ahol a tesztvégrehajtás sorrendje a követelményekhez tartozó tesztesetek prioritásaitól függ. A követelmény prioritásait az érdekelt felek határozzák meg. Elsőként azokat a teszteseteket hajtják végre, amelyek a legfontosabb követelményekhez kapcsolódnak.

Ideális esetben a teszteseteket prioritási szintjük alapján kellene futtatni, például az előzőleg említett priorizálási stratégiák egyikét használva. Azonban ez a gyakorlat nem működhet, ha a teszteseteknek vagy a tesztelt funkcióknak függőségei vannak. Ha egy magasabb prioritású teszteset függ egy alacsonyabb prioritású tesztesettől, akkor az alacsonyabb prioritású tesztesetet kell először végrehajtani.

A tesztvégrehajtás sorrendjének megállapításánál figyelembe kell venni az erőforrások elérhetőségét is. Például azokat a szükséges teszteszközöket, tesztkörnyezeteket vagy embereket, akik csak egy adott időablakban érhetők el.

5.1.6 Tesztpiramis

A tesztpiramis egy modell, amely megmutatja, hogy a különböző tesztek más-más szemcsézettségűek lehetnek. A tesztpiramis modell támogatja a csapatot a tesztautomatizálásban és a tesztráfordítás kiosztásában azáltal, hogy bemutatja, hogyan támogatják a különböző célokat a különböző tesztautomatizálási szintek. A piramis rétegek tesztek csoportját képviselik. Minél magasabb a réteg, annál

alacsonyabb a tesztszemcsézettség, a tesztelkülönítés és a tesztvégrehajtási idő. Az alsó rétegben lévő tesztek kicsik, elszigeteltek, gyorsak és a funkcionalitás egy kis részét ellenőrzik, ezért általában sok szükséges belőlük egy elfogadható lefedettség eléréséhez. A felső réteg bonyolult, magas szintű, végponttól végpontig tartó teszteket képvisel. Ezek a magas szintű tesztek általában lassabbak, mint az alsó rétegekben lévő tesztek, és általában a funkcionalitás nagy részét ellenőrzik, így általában csak néhányra van szükség az elfogadható lefedettség eléréséhez. A rétegek számossága és elnevezése eltérő lehet. Például az eredeti tesztpiramis modell (Cohn 2009) három réteget határoz meg: "egységtesztek", "szolgáltatástesztek" és "felhasználói felület tesztek". Egy másik népszerű modell egység (komponens) teszteket, integrációs (komponensintegrációs) teszteket és végponttól végpontig tartó teszteket határoz meg. De más tesztszinteket (lásd a 2.2.1. fejezetet) is használhatunk.

5.1.7 Tesztelési kvadránsok

A tesztelési kvadránsok, melyeket Brian Marick határozott meg (Marick 2003, Crispin 2008), az agilis szoftverfejlesztés során a megfelelő teszttípusokkal, tevékenységekkel, teszttechnikákkal és munkatermékekkel csoportosítják a tesztszinteket. A modell segíti a tesztmenedzsmentet abban, hogy megjelenítse az összes megfelelő teszttípust és tesztszintet a szoftverfejlesztési életciklusban belül, és abban is, hogy megértsük, egyes teszttípusok relevánsabbak bizonyos tesztszinteken, mint mások. Ez a modell emellett lehetőséget nyújt a tesztek típusainak differenciálására és leírására az összes érdekelt fél számára, ideértve a fejlesztőket, tesztelőket és az üzleti képviselőket is.

Ebben a modellben a tesztek lehetnek üzleti szemléletűek vagy technológiai szemléletűek. A tesztek támogathatják a csapatot (azaz irányíthatják a fejlesztést) vagy értékelhetik a terméket (azaz mérik annak viselkedését az elvárásokhoz képest). Ennek a két szemléletnek a kombinációja határozza meg a négy kvadránst:

- Q1 kvadráns (technológiai szemléletű, a csapatot támogatja). Ebben a kvadránsban találhatók a komponens- és komponensintegrációs tesztek. Ezeket a teszteket automatizálni kell, befoglalva a CI folyamatba.
- Q2 kvadráns (üzleti szemléletű, a csapatot támogatja). Ebben a kvadránsban találhatók a funkcionális tesztek, példák, felhasználói történet tesztek, felhasználói tapasztalat prototípusok, API tesztelés és szimulációk. Ezek a tesztek az elfogadási feltételeket ellenőrzik, és lehetnek manuálisak vagy automatizáltak.
- Q3 kvadráns (üzleti szemléletű, értékeli a terméket). Ebben a kvadránsban található a felderítő tesztelés, használhatósági tesztelés és felhasználói elfogadási tesztelés. Ezek a tesztek a felhasználókra összpontosítanak, és gyakran manuálisak.
- Q4 kvadráns (technológiai szemléletű, értékeli a terméket). Ebben a kvadránsban találhatók a smoke tesztek és a nemfunkcionális tesztek (a használhatósági teszteket kivéve). Ezek a tesztek gyakran automatizáltak.

5.2 Kockázatmenedzsment

A szervezetek számos belső és külső tényezővel szembesülnek, amelyek miatt bizonytalan, hogy mikor és hogyan érik el céljaikat (ISO 31000). A kockázatmenedzsment lehetővé teszi a szervezetek számára, hogy növeljék a céljaik elérésének valószínűségét, javítsák termékeik minőségét, és növeljék az érdekelt felek bizalmát.

A fő kockázatmenedzsment tevékenységek magukban foglalják:

- Kockázatelemzés (kockázatazonosításból és kockázatértékelésből áll; lásd az 5.2.3. fejezetet)
- Kockázatirányítás (kockázatmérséklésből és kockázatfelügyeletből áll; lásd az 5.2.4. fejezetet)

Azt a tesztmegközelítést, amelyben a teszttevékenységeket a kockázatelemzés és kockázatirányítás alapján választják ki, valamint priorizálják és kezelik, kockázatalapú tesztelésnek hívják.

5.2.1 Kockázat meghatározása és kockázat tulajdonságai

A kockázat egy olyan lehetséges esemény, veszély, fenyegetés vagy helyzet, amelynek bekövetkezése káros hatást eredményez. Egy kockázatot két tényező jellemez:

- Kockázatvalószínűség: a kockázat bekövetkezésének valószínűsége (nullánál nagyobb és egynél kisebb)
- Kockázat hatása (kár): a kockázat bekövetkezésének következményei

Ez a két tényező együttesen fejezi ki a kockázati szintet, amely a kockázat mértékét méri. Minél magasabb a kockázati szint, annál fontosabb annak kezelése.

5.2.2 Projektkockázatok és termékkockázatok

Szoftvertesztelés során általában két típusú kockázattal kell számolni: projektkockázatokkal és termékkockázatokkal.

Projektkockázatok: a projekt menedzsmentjéhez és irányításához kapcsolódnak. Ezek a kockázatok magukban foglalják:

- Szervezeti problémákat (pl. késés a munkatermék szállításában, pontatlan becslések, költségcsökkentés)
- Emberi problémákat (pl. hiányos készségek, konfliktusok, kommunikációs problémák, személyzet hiánya)
- Technikai problémákat (pl. hatókör növekedés, gyenge eszköztámogatás)
- Beszállítói problémákat (pl. harmadik fél szállítási hibája, a támogató cég csődje)

Amikor a projektkockázatok bekövetkeznek, hatással lehetnek a projekt ütemtervére, költségvetésére vagy hatókörére, ami befolyásolja a projekt céljainak elérését.

Termékkockázatok: a termék minőségjellemzőihez kapcsolódnak (pl. ahogyan az ISO 25010 minőségi modellben szerepel). A termékkockázatok példái közé tartozik a hiányzó vagy helytelen funkcionalitás, számítási hibák, futási problémák, rossz architektúra, nem megfelelő algoritmusok, elégtelen válaszidő, gyenge felhasználói élmény és biztonsági sebezhetőségek. Amikor a termékkockázatok bekövetkeznek, különböző negatív következményekkel járhatnak, ideértve:

- Felhasználói elégedetlenséget
- Bevétel, bizalom és hírnév elvesztését
- Harmadik félnek okozott károkat
- Magas karbantartási költségeket és az ügyfélszolgálat túlterheltségét
- Büntetőjogi büntetéseket
- Szélsőséges esetekben fizikai károkat, sérüléseket vagy akár halált is.

5.2.3 Termékkockázat-elemzés

Tesztelési szempontból a termékkockázat-elemzés célja, hogy felhívja a figyelmet a termékkockázatra a tesztelési ráfordításokra való összpontosításával annak érdekében, hogy minimalizáljuk a termékkockázatnak megmaradó szintjét. Ideális esetben a termékkockázat-elemzés már a szoftvertervezési életciklus korai szakaszában elkezdődik.

A termékkockázat-elemzés két részből áll: a kockázatazonosításból és kockázatértékelésből. A kockázatazonosítás a kockázatok teljes körű listájának létrehozásáról szól. Az érdekelt felek különböző technikák és eszközök segítségével azonosíthatják a kockázatokat, például ötletelés, workshopok, interjúk vagy ok-okozati diagramok segítségével. A kockázatértékelés magában foglalja az azonosított kockázatok kategorizálását, a kockázat valószínűségének, hatásának és szintjének meghatározását, priorizálást és megoldási módok javaslatát. A kategorizálás segít a mérséklő intézkedések hozzárendelésében, mert általában az ugyanabba a kategóriába tartozó kockázatok hasonló megközelítéssel mérsékelhetők.

A kockázatértékelés mennyiségi vagy minőségi megközelítést, vagy ezek kombinációját alkalmazhatja. A mennyiségi megközelítésben a kockázati szintet a kockázat valószínűségének és hatásának szorzataként számolják ki. Minőségi megközelítésben a kockázati szintet kockázati mátrix segítségével lehet meghatározni.

A termékkockázat-elemzés befolyásolhatja a tesztelés alaposságát és hatókörét. Az eredményeit az alábbi célokra használják:

- Meghatározza a végrehajtandó tesztelés hatókörét
- Meghatározza a konkrét tesztszinteket és javasolja a végrehajtandó teszttípusokat
- Meghatározza a végrehajtandó teszttechnikákat és az elérni kívánt lefedettséget
- Felméri az egyes feladatokhoz szükséges tesztráfordítást
- Priorizálja a tesztelést, hogy minél korábban megtalálja a kritikus hibákat
- Meghatározza, hogy a kockázat csökkentése érdekében tesztelésen kívüli tevékenységek alkalmazhatók-e.

5.2.4 Termékkockázat-irányítás

A termékkockázat-irányítás magában foglalja azon intézkedéseket, amelyeket a meghatározott és értékelt termékkockázatokra adnak. Tartalmazza a kockázatmérséklést és a kockázatfelügyeletet. A kockázatmérséklés magában foglalja a kockázatértékelésben javasolt intézkedések végrehajtását a kockázati szint csökkentése érdekében. A kockázatfelügyelet célja annak biztosítása, hogy a mérséklő intézkedések hatékonyak legyenek, további információkat szerezzenek a kockázatértékelés javításához, és azonosítsák az előbukkanó kockázatokat.

A termékkockázat-irányításának vonatkozásában, miután egy kockázatot elemeztek, több választlehetőség lehetséges a kockázatra, például kockázatmérséklés teszteléssel, kockázat elfogadás, kockázat átadás vagy tartalékterv (Veenendaal 2012). A termékkockázatok mérséklésére teszteléssel alkalmazható intézkedések a következők:

- Válassza ki az adott kockázattípushoz megfelelő szintű tapasztalattal és készségekkel rendelkező tesztelőket.
- Alkalmazzon megfelelő szintű tesztelési függetlenséget.
- Folytasson felülvizsgálatokat és végezzen statikus elemzést.
- Alkalmazzon megfelelő teszttechnikákat és lefedettségi szinteket.
- Alkalmazzon megfelelő teszttípusokat, amelyek az érintett minőségjellemzőket célozzák meg.

- Hajtson végre dinamikus tesztelést, ideértve a regressziós tesztelést is.

5.3 Tesztfelügyelet, tesztirányítás és tesztlezárás

A tesztfelügyelet a teszteléssel kapcsolatban való információgyűjtéssel foglalkozik. Ezt az információt a tesztelés előrehaladásának értékelésére és a teszt kilépési feltételek vagy a kilépési feltételekkel kapcsolatos tesztfeladatok teljesítésének mérésére használják, például, hogy elérték-e a termékkockázatok, követelmények vagy elfogadási feltételek lefedettségi céljait.

A tesztirányítás az információkat a tesztfelügyeletből használja fel, irányítási direktívák formájában nyújtva iránymutatást és szükséges korrekciós intézkedéseket a leghatékonyabb és legeredményesebb tesztelés eléréséhez. Példák az irányítási direktívákra:

- Tesztek prioritásainak újrarendezése, amikor egy azonosított kockázat problémává válik.
- Újraértékelés, hogy egy tesztelem megfelel-e a belépési feltételeknek vagy kilépési feltételeknek az átdolgozás miatt.
- A teszt ütemterv módosítása a tesztkörnyezet szállításának késése miatt.
- Új erőforrások hozzáadása, amikor és ahol szükséges.

A tesztlezárás adatokat gyűjt a teljesített teszttevékenységekből, hogy összegezze a tapasztalatokat, a tesztvert és bármilyen más releváns információt. A tesztlezárási tevékenységek projekt mérföldkövekhez kapcsolódnak, például amikor lezáródik egy tesztszint, befejeződik egy agilis iteráció, lezáródik egy tesztprojekt (vagy megszakítják), egy szoftverrendszer kiadásra kerül, vagy egy karbantartási kiadás befejeződik.

5.3.1 Tesztelésben használt metrikák

A tesztmetrikákat azért gyűjtjük, hogy lássuk az előrehaladást az ütemtervhez és költségvetéshez képest, lássuk a teszt tárgyának jelenlegi minőségét, illetve a teszttevékenységek hatékonyságát a végső célokkal, vagy egy iteráció céljaival kapcsolatban. A tesztfelügyelet számos metrikát gyűjt, hogy támogassa a tesztirányítást és a tesztlezárást.

Gyakori tesztmetrikák:

- Projektelőrehaladási metrikák (pl. feladatlezárás, erőforrásfelhasználás, tesztráfordítás)
- Tesztelőrehaladási metrikák (pl. teszteset-megvalósítás előrehaladás, tesztkörnyezet előkészítési előrehaladás, futtatott / nem futtatott tesztesetek száma, sikeres / bukott tesztfuttatások, tesztvégrehajtási idő)
- Termékminőségi metrikák (pl. rendelkezésre állás, válaszidő, átlagos idő a meghibásodásig)
- Hiba metrikák (pl. talált/javított hibák száma és prioritása, hibasűrűség, hibamegtalálási arány)
- Kockázati metrikák (pl. maradék kockázatszint)
- Lefedettségi metrikák (pl. követelmény-lefedettség, kódlefedettség)
- Költség metrikák (pl. tesztelés költsége, minőség szervezeti költsége)

5.3.2 Tesztjelentések célja, tartalma és célközönsége

A tesztjelentések összefoglalják a tesztekkel kapcsolatos információkat. Ezek az információk a tesztelés során, illetve a tesztelés után is kommunikálásra kerülnek. A tesztelőrehaladási jelentések támogatják a tesztelés folyamatos irányítását, és elegendő információt kell nyújtaniuk a tesztütemterv, erőforrások vagy

teszterv módosításához, amikor az ilyen változások a tervtől való eltérés vagy megváltozott körülmények miatt szükségesek. Az összefoglaló tesztjelentések összefoglalják a tesztelés egy adott szakaszát (pl. tesztszint, tesztciklus, iteráció) és információt adhatnak a későbbi teszteléshez.

A tesztfelügyelet és irányítás során a tesztcsapat tesztelőrehaladási jelentéseket készít az érdekelt felek számára, hogy tájékoztassák őket. A tesztelőrehaladási jelentéseket általában rendszeres időközönként készítik (pl. naponta, hetente stb.) és magukban foglalják:

- Tesztidőszak
- Tesztelőrehaladás (pl. előrébb vagy hátrébb van az ütemtervhez képest), beleértve a jelentősebb eltéréseket
- Akadályok a tesztelés során, és azok megoldásai
- Tesztmetrikák (lásd az 5.3.1. fejezetet példákért)
- A tesztelési időszakban felmerült új és megváltozott kockázatok
- A következő időszakra tervezett tesztelés

Az összefoglaló tesztjelentés tesztlezáráskor készül, amikor egy projekt, tesztszint vagy teszttípus lezárul, és ideális esetben a kilépési feltételeknek is megfelel. Ez a jelentés a tesztelőrehaladási jelentéseken és más adatokon alapszik. A tipikus összefoglaló tesztjelentések magukban foglalják:

- Teszt összefoglalása
- A tesztelés és a termék minőségének értékelése az eredeti teszterv alapján (azaz a tesztcélok és kilépési feltételek)
- Eltérések a teszttervtől (pl. a tervezett ütemterv, időtartam és ráfordítás közötti különbségek)
- Tesztelési akadályok és azok megoldásai
- Tesztmetrikák a tesztelőrehaladási jelentések alapján
- Még nem mérsékelt kockázatok, javítatlan hibák
- A teszteléssel kapcsolatos tanulságok

A különböző célcsoportok különböző információkat igényelnek a jelentésekben, és befolyásolják a jelentések formális jellegét és gyakoriságát. A tesztelőrehaladásról való jelentés a csapat belső tagjai számára többnyire gyakori és informális, míg a tesztelésről való jelentés egy lezárt projekt esetén egy meghatározott sablont követ, és csak egyszer történik meg. Az ISO/IEC/IEEE 29119-3 szabvány sablonokat és példákat tartalmaz a tesztelőrehaladási jelentések (tesztstátuszjelentések) és az összefoglaló tesztjelentések számára.

5.3.3 Tesztelés státuszának kommunikálása

A tesztstátusz kommunikációjának legjobb módja függ a tesztmenedzsment különböző szempontjaitól, a szervezet tesztstratégiáitól, a szabályozói szabványoktól, illetve, önszerveződő csapatok esetén (lásd az 1.5.2. fejezetet) a csapat saját működésétől. Az opciók magukban foglalják:

- Szóbeli kommunikáció a csapattagokkal és más érdekelt felekkel
- Áttekintőtáblák (pl. CI/CD áttekintőtáblák, feladattáblák és burn-down chartok)
- Elektronikus kommunikációs csatornák (pl. e-mail, csevegés)
- Online dokumentáció
- Formális tesztjelentések (lásd az 5.3.2. fejezet)

Ezek közül egy vagy több lehetőséget lehet használni. A formálisabb kommunikáció esetleg megfelelőbb lehet az elosztott csapatok számára, ahol a közvetlen személyes kommunikáció nem mindig lehetséges a földrajzi távolság vagy időeltérés miatt. Általában különböző érdekelt felek számára különböző típusú információk az érdekesek, így a kommunikációt is ennek megfelelően szükséges kialakítani.

5.4 Konfigurációmenedzsment

A tesztelés során a konfigurációmenedzsment (configuration management, CM) egy diszciplínát nyújt a teszttervek, tesztstratégiák, tesztfeltételek, tesztesetek, tesztszkriptek, teszteredmények, tesztnaplók, tesztjelentések és egyéb munkatermékek konfigurációelemekként való azonosításához, ellenőrzéséhez és nyomon követéséhez.

Egy összetett konfigurációelem (pl. egy tesztkörnyezet) esetén a CM rögzíti az azt alkotóó elemeket, azok kapcsolatait és verzióit. Ha a konfigurációelem jóváhagyást kap a teszteléshez, akkor viszonyítási alap lesz, és csak egy formális változáskezelési folyamaton keresztül módosítható.

A konfigurációmenedzsment rögzíti a megváltozott konfigurációelemeket, amikor új viszonyítási alapot hoznak létre. Lehetőség van visszatérni egy korábbi viszonyítási alaphoz a korábbi teszteredmények reprodukálása végett.

A tesztelést megfelelően támogatva a CM a következőket biztosítja:

- Az összes konfigurációelem, beleértve a tesztelemeket (amelyek a teszt tárgyának egyes részei), egyedileg azonosított, verziókezelt, változások szempontjából nyomonkövetett, és kapcsolatban áll más konfigurációelemekkel, hogy a nyomkövetés fenntartható legyen a tesztfolyamat teljes időtartama alatt.
- Azonosított dokumentációkat és szoftverelemeket világosan hivatkozzák a tesztdokumentációban.

A folyamatos integráció, folyamatos szállítás, folyamatos telepítés és a hozzájuk kapcsolódó tesztelés általában az automatizált DevOps folyamatlánc részeként van megvalósítva (lásd a 2.1.4. fejezet), amelybe általában az automatizált CM is tartozik.

5.5 Hibamenedzsment

Mivel az egyik fő tesztcél a hibák megtalálása, elengedhetetlen egy kialakult hibamenedzsment folyamat. Bár itt "hibákról" beszélünk, a bejelentett anomáliák valódi hibává vagy valami mássá (pl. fals pozitív, változtatáskérés) válhatnak - ez a hibajelentésekkel való foglalkozás folyamatában oldódik meg. Anomáliák a szoftverfejlesztési életciklus bármelyik fázisában bejelenthetők. Az anomália típusa az életcikluson múlik. A hibamenedzsment folyamata legalább az egyedi anomáliák kezelésére szolgáló, felfedezésüktől a lezárásukig tartó munkafolyamatot, és az osztályozásukhoz való szabályokat magába foglalja. A munkafolyamat jellemzően abból áll, hogy naplózzuk a jelentett anomáliákat, elemezzük és osztályozzuk őket, döntünk a sorsukról (például, hogy megjavítjuk, vagy úgy hagyjuk őket), és végül lezárjuk a hibajelentést. A folyamatot minden érdekelt félnek követnie kell. A hibákat statikus teszteléssel (különösen statikus elemzéssel) is ajánlott kezelni.

A tipikus hibajelentések a következő célokat szolgálják:

- Azoknak, akik felelősek a bejelentett hibák kezeléséért és megoldásáért, elegendő információt nyújtani a probléma megoldásához
- Eszközt biztosítani a munkatermék minőségének nyomonkövetéséhez
- Ötletekkel szolgálni a fejlesztési és tesztelési folyamat fejlesztéséhez

Egy dinamikus tesztelés során rögzített hibajelentés tipikusan az alábbiakat tartalmazza:

- Egyedi azonosító
- Cím, rövid összefoglalóval a bejelentett anomáliáról
- Dátum, amikor az anomáliát észlelték, kibocsátó szervezet, szerző, beleértve a szerepüket
- A teszt tárgyának és a tesztkörnyezetnek az azonosítása
- A hiba környezete (pl. futtatott teszteset, folyamatban lévő teszttevékenység, szoftverfejlesztési életciklus fázis és egyéb releváns információk, például a használt teszttechnika, ellenőrzőlista vagy tesztadatok)
- A hiba reprodukálását és megoldását lehetővé tevő leírás, beleértve a lépéseket, amelyekkel észlelték az anomáliát, és minden releváns naplófájlt, adatbázis-mentést, képernyőképet vagy -felvételt
- Elvárt és tényleges eredmények
- A hiba súlyossága (hatás mértéke) az érdekelt felek vagy követelmények szempontjából
- A hiba javításának prioritása
- A hiba állapota (pl. nyitott, elhalasztott, duplikált, megoldásra váró, ellenőrző tesztelésre váró, újranyitott, lezárt, visszautasított)
- Hivatkozások (pl. a tesztesetre)

Hibamenedzsment eszközök használatával ezen adatok egy része automatikusan rögzítésre kerülhet (pl. azonosító, dátum, szerző és kezdeti állapot). A hibajelentés tartalmára az ISO szabványban (ISO/IEC/IEEE 29119-3) található példa, amely a hibajelentésre, mint incidensjelentésre hivatkozik.

6. Teszteszközök 20 perc

Kulcsszavak

tesztautomatizálás

Tanulási célok a 6. fejezethez:

6.1 Eszköztámogatás a teszteléshez

AK-6.1.1 (K2) Magyarázza el, hogy a különböző típusú teszteszközök hogyan támogatják a tesztelést

6.2 A tesztautomatizálás előnyei és kockázatai

AK-6.1.2 (K1) Idézze fel a tesztautomatizálás előnyeit és kockázatait

6.1 Eszköztámogatás a teszteléshez

A teszteszközök számos teszttevékenységet támogatnak és könnyítenek meg. Ilyenek például, de nem kizárólag:

- Menedzsmenteszközök - a tesztfolyamat hatékonyságának növelése a szoftverfejlesztési életciklus, a követelmények, a tesztek, a hibák és a konfiguráció kezelésének megkönnyítésével
- Statikus tesztelési eszközök – a tesztelő támogatása a felülvizsgálatok és statikus elemzés elvégzésében
- Műszaki teszttervezés és tesztmegvalósítás eszközei – a tesztesetek, tesztadatok és teszteljárások generálásának megkönnyítése
- Tesztvégrehajtási és lefedettségi eszközök – az automatizált tesztvégrehajtás és a tesztlefedettség mérésének lehetővé tétele
- Nemfunkcionális tesztelés eszközei – lehetővé teszik a tesztelő számára a manuálisan nehezen vagy egyáltalán nem elvégezhető nemfunkcionális tesztelés elvégzését
- DevOps eszközök – a DevOps szállítási folyamatlánc támogatása, munkafolyamatok nyomonkövetése, automatizált buildelési folyamat(ok), CI/CD
- Együttműködési eszközök – a kommunikáció megkönnyítése
- A skálázhatóságot és a telepítés szabványosítását támogató eszközök (pl. virtuális gépek, konténerizációs eszközök)
- Bármilyen más, a tesztelést segítő eszköz (pl. egy táblázatkezelő is teszteszköznek tekinthető, ha teszteléshez használják)

6.2 A tesztautomatizálás előnyei és kockázatai

Egy eszköz beszerzése önmagában nem garantálja a sikert. Minden új eszköz erőfeszítést igényel a valódi és tartós előnyök eléréséhez (pl. az eszköz bevezetéséhez, karbantartásához és a képzéshez). Vannak kockázatok is, amelyeket elemezni és csökkenteni kell.

A tesztautomatizálás használatának többek között a következő előnyei lehetnek:

- Időmegtakarítás az ismétlődő manuális munka csökkentésével (pl. regressziós tesztek végrehajtása, ugyanazon tesztadatok újbóli bevitele, a várt eredmények és a tényleges eredmények összehasonlítása, valamint a kódolási szabványok szerinti ellenőrzés)
- Az egyszerű emberi eredetű hibák megelőzése a nagyobb következetesség és megismételhetőség révén (pl. a teszteket következetesen a követelményekből vezetik le, a tesztadatokat szisztematikusan hozzák létre és a teszteket egy eszközzel ugyanabban a sorrendben és ugyanolyan gyakorisággal hajtják végre)
- Objektívebb értékelés (pl. lefedettség) és az emberek számára túlságosan bonyolultan előállítható mérőszámok biztosítása
- Könnyebb hozzáférés a teszteléssel kapcsolatos információkhoz a tesztmenedzsment és a tesztjelentések készítésének támogatása érdekében (pl. statisztikák, grafikonok és összesített adatok a tesztelés előrehaladásáról, a hibaarányokról és a tesztvégrehajtás időtartamáról)
- Csökkentett tesztvégrehajtási idő a hibák korábbi felismerése, a gyorsabb visszajelzés és a gyorsabb piacra jutás érdekében

- Több idő áll a tesztelők rendelkezésére, hogy új, mélyebb és hatékonyabb teszteket tervezzenek

A tesztautomatizálás használatának többek között a következő kockázatai lehetnek:

- Irreális elvárások egy eszköz előnyeivel kapcsolatban (beleértve a funkcionalitást és a használat könnyűségét)
- Az eszköz bevezetéséhez, a tesztszkriptek karbantartásához és a meglévő manuális tesztelési folyamat megváltoztatásához szükséges idő, költség és erőfeszítés pontatlan becslése
- Teszteszköz használata, amikor a manuális tesztelés megfelelőbb
- Túlzottan támaszkodni az eszközre, például figyelmen kívül hagyni az emberi kritikus gondolkodás szükségességét
- Függőség az eszköz gyártójától, aki esetleg visszavonul az üzlettől, visszavonja vagy eladja az eszközt egy másik gyártónak, vagy gyenge támogatást nyújt (pl. a kérdésekre adott válaszok, frissítések és hibajavítások tekintetében)
- Nyílt forráskódú szoftver használata, amelyet lehet, hogy magára hagynak, azaz nem állnak majd rendelkezésre további frissítések, vagy úgy fejlesztenek, hogy belső összetevőit gyakran frissíteni kell
- Az automatizálási eszköz nem kompatibilis a fejlesztési platformmal
- Olyan eszköz választása, amely nem felel meg a szabályozói követelményeknek és/vagy a biztonsági szabványoknak

7. Irodalomjegyzék

Szabványok

ISO/IEC/IEEE 29119-1 (2022) Software and systems engineering - Software testing - Part 1: General Concepts

ISO/IEC/IEEE 29119-2 (2021) Software and systems engineering - Software testing - Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2021) Software and systems engineering - Software testing - Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2021) Software and systems engineering - Software testing - Part 4: Test techniques

ISO/IEC 25010, (2011) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models

ISO/IEC 20246: (2017) Software and systems engineering — Work product reviews

ISO/IEC/IEEE 14764:2022 – Software engineering – Software life cycle processes – Maintenance

ISO 31000 (2018) Risk management – Principles and guidelines

Könyvek

Adzic, G. (2009) Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing, Neuri Limited

Ammann, P. and Offutt, J. (2016) Introduction to Software Testing (2e), Cambridge University Press

Andrews, M. and Whittaker, J. (2006) How to Break Web Software: Functional and Security Testing of Web Applications and Web Services, Addison-Wesley Professional

Beck, K. (2003) Test Driven Development: By Example, Addison-Wesley

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

Boehm, B. (1981) Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ

Buxton, J.N. and Randell B., eds (1970), Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27–31 October 1969, p. 16

Chelimsky, D. et al. (2010) The Rspec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends, The Pragmatic Bookshelf: Raleigh, NC

Cohn, M. (2009) Succeeding with Agile: Software Development Using Scrum, Addison-Wesley

Copeland, L. (2004) A Practitioner's Guide to Software Test Design, Artech House: Norwood MA

Craig, R. and Jaskiel, S. (2002) Systematic Software Testing, Artech House: Norwood MA

Crispin, L. and Gregory, J. (2008) Agile Testing: A Practical Guide for Testers and Agile Teams, Pearson Education: Boston MA

Forgács, I., and Kovács, A. (2019) Practical Test Design: Selection of traditional and automated test design techniques, BCS, The Chartered Institute for IT

Gawande A. (2009) The Checklist Manifesto: How to Get Things Right, New York, NY: Metropolitan Books

Gärtner, M. (2011), ATDD by Example: A Practical Guide to Acceptance Test-Driven Development, Pearson Education: Boston MA

Gilb, T., Graham, D. (1993) Software Inspection, Addison Wesley

Hendrickson, E. (2013) Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing, The Pragmatic Programmers

Hetzel, B. (1988) The Complete Guide to Software Testing, 2nd ed., John Wiley and Sons

Jeffries, R., Anderson, A., Hendrickson, C. (2000) Extreme Programming Installed, Addison-Wesley Professional

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

Kan, S. (2003) Metrics and Models in Software Quality Engineering, 2nd ed., Addison-Wesley

Kaner, C., Falk, J., and Nguyen, H.Q. (1999) Testing Computer Software, 2nd ed., Wiley

Kaner, C., Bach, J., and Pettichord, B. (2011) Lessons Learned in Software Testing: A Context-Driven Approach, 1st ed., Wiley

Kim, G., Humble, J., Debois, P. and Willis, J. (2016) The DevOps Handbook, Portland, OR

Koomen, T., van der Aalst, L., Broekman, B. and Vroon, M. (2006) TMap Next for result-driven testing, UTN Publishers, The Netherlands

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

O'Regan, G. (2019) Concise Guide to Software Testing, Springer Nature Switzerland

Pressman, R.S. (2019) Software Engineering. A Practitioner's Approach, 9th ed., McGraw Hill

Roman, A. (2018) Thinking-Driven Testing. The Most Reasonable Approach to Quality Control, Springer Nature Switzerland

Van Veenendaal, E (ed.) (2012) Practical Risk-Based Testing, The PRISMA Approach, UTN Publishers: The Netherlands

Watson, A.H., Wallace, D.R. and McCabe, T.J. (1996) Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, U.S. Dept. of Commerce, Technology Administration, NIST

Westfall, L. (2009) The Certified Software Quality Engineer Handbook, ASQ Quality Press

Whittaker, J. (2002) How to Break Software: A Practical Guide to Testing, Pearson

Whittaker, J. (2009) Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design, Addison Wesley

Whittaker, J. and Thompson, H. (2003) How to Break Software Security, Addison Wesley

Wiegers, K. (2001) Peer Reviews in Software: A Practical Guide, Addison-Wesley Professional

Cikkek és weboldalak

Brykczynski, B. (1999) "A survey of software inspection checklists," ACM SIGSOFT Software Engineering Notes, 24(1), pp. 82-89

Enders, A. (1975) "An Analysis of Errors and Their Causes in System Programs," IEEE Transactions on Software Engineering 1(2), pp. 140-149

Manna, Z., Waldinger, R. (1978) "The logic of computer programming," IEEE Transactions on Software Engineering 4(3), pp. 199-229

Marick, B. (2003) Exploration through Example, http://www.exampler.com/old-blog/2003/08/21.1.html#agile-testing-project-1

Nielsen, J. (1994) “Enhancing the explanatory power of usability heuristics,” Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, ACM Press, pp. 152–158

Salman. I. (2016) “Cognitive biases in software quality and testing,” Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16), ACM, pp. 823-826.

Wake, B. (2003) “INVEST in Good Stories, and SMART Tasks,” https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

8. „A” függelék – Tanulási Célok/Kognitív Tudásszintek

A következő tanulási célkitűzések vannak meghatározva erre a tantervre. Minden egyes téma a tantervben az adott témához tartozó tanulási célok szerint képezi a vizsga tárgyát. A tanulási célok az alábbiakban felsorolt kognitív tudásszintjének megfelelő cselekvési igével kezdődnek.

1. szint: Emlékezés (K1)

A jelölt képes felismerni és felidézni a fogalmat vagy koncepciót.

Kulcsszavak: azonosítsa, idézze fel, emlékezzen, ismerje fel

Példák:

- „Azonosítsa a tipikus tesztcélokat.”
- „Idézze fel a tesztpiramis fogalmait.”
- „Ismerje fel, hogy egy tesztelő milyen értékekkel járul hozzá az iteráció- és kiadástervezéshez”

2. szint: Megértés (K2)

A jelölt képes az egyes témákhoz tartozó állítások okát és magyarázatát kiválasztani, képes összegezni, összehasonlítani, osztályozni, csoportosítani és a tesztelési koncepciót példákkal alátámasztani.

Kulcsszavak: osztályozza, hasonlítsa össze, állítsa szembe, különböztesse meg, szemléltesse, magyarázza el, adjon példákat, értelmezze, foglalja össze

Példák:

- „Osztályozza az elfogadási feltételek megírásának különböző lehetőségeit.”
- „Hasonlítsa össze a különböző szerepeket a tesztelésben” (keressen hasonlóságokat, különbségeket vagy mindkettőt).
- „Különböztesse meg a projektkockázatokat és a termékkockázatokat” (lehetővé teszi a fogalmak megkülönböztetését).
- „Szemléltesse egy teszterv célját és tartalmát.”
- „Magyarázza el a kontextus hatását a vizsgálati folyamatra.”
- „Foglalja össze a felülvizsgálati folyamat tevékenységeit.”

3. szint: Alkalmazás (K3)

A jelölt képes a koncepció, vagy a technika kiválasztására, valamint helyes alkalmazására egy adott környezetben.

Kulcsszavak: alkalmazzon, valósítsa meg, készítsen, használjon

Példák:

- „Alkalmazzon teszteset-priorizálást” (eljárásra, technikára, folyamatra, algoritmusra stb. kell utalnia).
- „Készítsen hibajelentést.”
- „Használjon határérték elemzést tesztesetek származtatásához."

Hivatkozások a tanulási célok kognitív tudásszintjeihez:

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching

Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon

9. „B” függelék – Az üzleti eredmények nyomonkövethetőségi mátrixa tanulási célokkal

Ez a szakasz felsorolja az üzleti eredményekhez kapcsolódó alapszintű tanulási célokat, valamint az alapszintű üzleti eredmények és az alapszintű tanulási célok közötti nyomonkövethetőséget.

Üzleti eredmények: Alapszintű tanterv		FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
BO1	Megérti, mi az a tesztelés és miért előnyös az.	6													
BO2	Megérti a szoftvertesztelés alapvető fogalmait.		22												
BO3	A tesztelési megközelítést és tevékenységeket a tesztelési kontextustól függően azonosítja és hajtja végre.			6											
BO4	Értékeli és javítja a dokumentáció minőségét.				9										
BO5	Növeli a tesztelés hatásosságát és hatékonyságát.					20									
BO6	A tesztelési folyamatot a szoftverfejlesztési életciklushoz igazítja.						6								
BO7	Megérti a tesztmenedzsment alapelveit.							6							
BO8	Világos és érthető hibajelentéseket ír és fogalmaz meg.								1						
BO9	Megérti a teszteléssel kapcsolatos prioritásokat és ráfordításokat befolyásoló tényezőket.									7					
BO10	Egy többfunkciós csapat részeként dolgozzon.										8				
BO11	Ismeri a tesztautomatizálással kapcsolatos kockázatokat és előnyöket.											1			
BO12	Azonosítja a teszteléshez szükséges alapvető készségeket.												5		
BO13	Megérti a kockázat tesztelésre gyakorolt hatását.													4	
BO14	Hatékonyan jelent a tesztelés haladásáról és minőségéről.														4

Fejezet/ szakasz/ alszakasz	Tanulási célok	K-szint	ÜZLETI EREDMÉNYEK													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
1. Fejezet	**A tesztelés alapjai**															
1.1	**Mi a tesztelés?**															
1.1.1	Azonosítson tipikus tesztcélokat	K1	X													
1.1.2	Különböztesse meg a tesztelést és a hibakeresést	K2		X												
1.2	**Miért szükséges a tesztelés?**															
1.2.1	Szemléltesse a tesztelés szükségességét	K2	X													
1.2.2	Idézze fel a tesztelés és a minőségbiztosítás közötti kapcsolatot	K2		X												
1.2.3	Tegyen különbséget a kiváltó ok, az emberi eredetű hiba, hiba és meghibásodás között	K2		X												
1.3	**Tesztelési alapelvek**															

1.3.1	Ismertesse az általános tesztelési alapelveket (A hét tesztelési alapelv)	K2		X											
1.4	**Teszttevékenységek, tesztver és teszt szerepkörök**														
1.4.1	Foglalja össze a különböző teszttevékenységeket és feladatokat	K2			X										
1.4.2	Ismertesse a kontextus tesztfolyamatra gyakorolt hatását	K2			X			X							
1.4.3	Tegyen különbséget a teszttevékenységeket támogató tesztverek között	K2			X										
1.4.4	Ismertesse a nyomonkövethetőség fenntartásának fontosságát	K2				X	X								
1.4.5	Hasonlítsa össze a tesztelés különböző szerepköreit	K2									X				
1.5	**Alapvető készségek és jó gyakorlatok a tesztelésben**														
1.5.1	Adjon példákat a teszteléshez szükséges készségekre	K2											X		
1.5.2	Idézze fel a teljes csapat megközelítés előnyeit	K1									X				
1.5.3	Tegyen különbséget a tesztelés függetlenségének előnyei és hátrányai között	K2			X										
2. Fejezet	**Tesztelés a szoftverfejlesztés életciklusán át**														
2.1	**Tesztelés a szoftverfejlesztési életciklus kontextusában**														
2.1.1	Ismertesse a kiválasztott szoftverfejlesztési életciklus tesztelésre gyakorolt hatását	K2					X								
2.1.2	Idézzen fel hasznos tesztelési gyakorlatokat, amelyek alkalmazhatóak minden szoftverfejlesztési életciklus esetén	K1					X								
2.1.3	Idézze fel a test-first megközelítés példáit a fejlesztésben	K1					X								
2.1.4	Foglalja össze, hogyan hathat a DevOps a tesztelésre	K2					X	X			X	X			
2.1.5	Ismertesse a shift left megközelítést	K2					X	X							
2.1.6	Ismertesse, hogyan használhatóak a visszatekintő megbeszélések a folyamatfejlesztés eszközeiként	K2					X					X			
2.2	**Tesztszintek és teszttípusok**														
2.2.1	Különböztesse meg az eltérő tesztszinteket	K2		X	X										
2.2.2	Különböztesse meg az eltérő teszttípusokat	K2		X											
2.2.3	Különböztesse meg az ellenőrző tesztelést és a regressziós tesztelést	K2		X											
2.3	**Karbantartási tesztelés**														
2.3.1	Foglalja össze a karbantartási tesztelést és az azt szükségessé tevő tényezőket	K2		X					X						
3. Fejezet	**Statikus tesztelés**														
3.1	**Statikus tesztelési alapok**														
3.1.1	Ismerje fel azokat a munkatermék típusokat, amelyek vizsgálhatók a különböző statikus tesztelési technikákkal	K1				X	X								
3.1.2	Ismertesse a statikus tesztelés előnyeit	K2	X			X	X								
3.1.3	Hasonlítsa össze a statikus és a dinamikus tesztelést és vizsgálja meg a különbségeket	K2				X	X								
3.2	**Visszajelzés és a felülvizsgálat folyamata**														
3.2.1	Azonosítsa az érdekelt felek számára végzett korai és gyakori visszajelzések előnyeit	K1	X			X					X				

3.2.2	Foglalja össze a felülvizsgálati folyamat egyes tevékenységeit	K2			X	X									
3.2.3	Idézze fel, hogy milyen felelősségek rendelhetők a főbb szerepekhez a felülvizsgálatok végrehajtása során	K1				X							X		
3.2.4	Hasonlítsa össze a különböző felülvizsgálat típusokat és vizsgálja meg a különbségeket	K2		X											
3.2.5	Idézze fel a sikeres felülvizsgálathoz hozzájáruló tényezőket	K1					X						X		
4. Fejezet	**Tesztelemzés és műszaki teszttervezés**														
4.1	**Teszttechnikák áttekintése**														
4.1.1	Tegyen különbséget a feketedoboz, fehérdoboz - és a tapasztalatalapú teszttechnikák között	K2		X											
4.2	**Feketedoboz-teszttechnikák**														
4.2.1	Alkalmazza az ekvivalenciapartícionálást tesztesetek származtatására	K3					X								
4.2.2	Alkalmazza a határérték-elemzést tesztesetek származtatására	K3					X								
4.2.3	Alkalmazza a döntési tábla tesztelést tesztesetek származtatására	K3					X								
4.2.4	Alkalmazza az állapotátmenet tesztelést tesztesetek származtatására	K3					X								
4.3	**Fehérdoboz-teszttechnikák**														
4.3.1	Magyarázza el az utasítástesztelést	K2		X											
4.3.2	Magyarázza el az elágazás tesztelést	K2		X											
4.3.3	Magyarázza el a fehérdoboz tesztelés értékét	K2	X	X											
4.4	**Tapasztalatalapú teszttechnikák**														
4.4.1	Magyarázza el a hibasejtést	K2		X											
4.4.2	Magyarázza el a felderítő tesztelést	K2		X											
4.4.3	Magyarázza el az ellenőrzőlista-alapú tesztelést	K2		X											
4.5	**Együttműködés-alapú tesztmegközelítések**														
4.5.1	Magyarázza el a fejlesztőkkel és üzleti képviselőkkel együttműködésben történő fejlesztői történetek írásának mikéntjét	K2				X					X				
4.5.2	Osztályozza az elfogadási feltételek írásának különböző lehetőségeit	K2									X				
4.5.3	Alkalmazza az elfogadásiteszt-vezérelt fejlesztést (ATDD) tesztesetek származtatására	K3					X								
5. Fejezet	**Tesztevékenységek menedzselése**														
5.1	**Teszttervezés**														
5.1.1	Szemléltesse a tesztterv célját és tartalmát	K2		X					X						
5.1.2	Ismerje fel, hogyan ad hozzá értéket a tesztelő az iteráció- és kiadástervezéshez	K1	X								X		X		
5.1.3	Hasonlítsa össze a belépési és kilépési feltételeket	K2				X		X							X
5.1.4	Használjon becslési technikákat a szükséges tesztráfordítás kiszámításához	K3							X		X				
5.1.5	Alkalmazzon teszteset-priorizálást	K3							X		X				
5.1.6	Idézze fel a tesztpiramis fogalmait	K1		X											

5.1.7	Foglalja össze a tesztelési kvadránsokat és azok kapcsolatát a tesztszintekkel és teszttípusokkal	K2		X						X						
5.2	**Kockázatmenedzsment**															
5.2.1	Azonosítsa a kockázati szintet a kockázat valószínűségének és hatásának felhasználásával	K1							X						X	
5.2.2	Különböztesse meg a projektkockázatokat és termékkockázatokat	K2		X											X	
5.2.3	Magyarázza el, hogyan befolyásolhatja a termékkockázat-elemzés a tesztelés alaposságát és hatókörét	K2					X			X					X	
5.2.4	Magyarázza el, milyen intézkedéseket lehet tenni az elemzett termékkockázatokra reagálva	K2		X			X								X	
5.3	**Tesztfelügyelet, tesztirányítás és tesztlezárás**															
5.3.1	Idézze fel a teszteléshez használt metrikákat	K1								X						X
5.3.2	Foglalja össze a tesztjelentések célját, tartalmát és célközönségét	K2					X			X						X
5.3.3	Szemléltesse, hogyan lehet kommunikálni a tesztelés státuszát	K2											X			X
5.4	**Konfigurációmenedzsment**															
5.4.1	Foglalja össze, hogyan támogatja a konfigurációmenedzsment a tesztelést	K2					X		X							
5.5	**Hibamenedzsment**															
5.5.1	Készítsen egy hibajelentést	K3		X						X						
6. Fejezet	**Teszteszközök**															
6.1	**Eszköztámogatás a teszteléshez**															
6.1.1	Magyarázza el, hogy a különböző típusú teszteszközök hogyan támogatják a tesztelést	K2					X									
6.2	**A tesztautomatizálás előnyei és kockázatai**															
6.2.1	Idézze fel a tesztautomatizálás előnyeit és kockázatait	K1					X					X				

10. „C” függelék – Kiadási megjegyzések

Az ISTQB® Foundation Syllabus v4.0 egy nagyfokú átdolgozás, amely az Alapszintű Tanterven (v3.1.1) és az Agile Tester 2014 Tanterven alapul. Emiatt nincsenek részletes kiadási megjegyzések fejezetenként és szakaszonként. Az alábbiakban azonban összefoglaljuk a főbb változtatásokat. Ezenkívül egy külön kiadási megjegyzések dokumentumban az ISTQB® nyomonkövethetőséget biztosít az Alapszintű Tanterv 3.1.1-es verziójában, az Agilis Tester Tanterv 2014-es verziójában szereplő tanulási célok (TC) és az új Alapszintű Tanterv 4.0 tanulási céljai között, amely megmutatja, hogy mely TC-k kerültek hozzáadásra, frissítésre vagy eltávolításra.

A tanterv írásakor (2022-2023) több, mint egymillióan tettek alapszintű vizsgát összesen több, mint 100 országban és több, mint 800 000-en szereztek tanúsítványt világszerte. Feltételezve, hogy a sikeres vizsgájuk érdekében mindegyikük olvasta az Alapszintű Tantervet, az Alapszintű Tanterv a valaha készült legolvasottabb dokumentum a szoftvertesztelés témakörében. Ez a nagyfokú átdolgozás ennek az örökségnek a tiszteletben tartásával készült, illetve, hogy növelje azt az értéket, melyet az ISTQB® nyújt majd a globális tesztelői közösségben további több százezer ember számára.

Ebben a verzióban minden Tanulási Cél úgy lett megszerkesztve, hogy atomi, és egy az egyben nyomonkövethető legyen a TC-k és a tanterv szakaszai közötti kapcsolat, így nincs olyan tartalom, hogy ne lenne hozzá kapcsolódó TC is. A cél az, hogy ezt a verziót könnyebben olvashatóvá, megérthetővé, megtanulhatóvá és lefordíthatóvá tegyük, a gyakorlati hasznosság növelésére, valamint a tudás és készségek közötti egyensúlyra összpontosítva.

Ez a nagyfokú átdolgozás a következő változtatásokat tartalmazza:

- A teljes tanterv méretének csökkentése. A tanterv nem egy tankönyv, hanem egy dokumentum, amely egy bevezető szoftvertesztelési kurzus alapelemeinek felvázolására szolgál, beleértve azt is, hogy milyen témákat és milyen szinten kell lefedni. Ezért különösen:
  - A legtöbb esetben a szöveg nem tartalmaz példákat. A példák és a gyakorlatok biztosítása a képzés során az oktató feladata
  - A „Tanterv írási ellenőrző listát” követi, amely a TC-khez maximális szövegméretet javasol minden K-szinten (K1 = max. 10 sor, K2 = max. 15 sor, K3 = max. 25 sor)
- A TC-k számának csökkentése a Foundation v3.1.1 és az Agile v2014 tananyaghoz képest
  - 14 K1 TC az FL v3.1.1 (15) és AT 2014 (6) 21 TC-hez képest
  - 42 K2 TC az FL v3.1.1 (40) és AT 2014 (13) 53 TC-hez képest
  - 8 K3 TC az FL v3.1.1 (7) és AT 2014 (8) 15 TC-hez képest
- Kiterjedtebb hivatkozások találhatók klasszikus és/vagy elismert könyvekre és cikkekre a szoftvertesztelésről és a kapcsolódó témákról
- Főbb változások az 1. fejezetben (A tesztelés alapjai)
  - A tesztelői készségekre vonatkozó rész kibővült és javításra került
  - A teljes csapat megközelítésről szóló rész (K1) hozzáadva
  - A tesztelés függetlenségéről szóló rész az 5. fejezetből az 1. fejezetbe került
- Főbb változások a 2. fejezetben (Tesztelés a szoftverfejlesztés életciklusán át)
  - A 2.1.1 és 2.1.2 szakaszok átírva és javítva, a megfelelő TC-k módosultak
  - Több hangsúlyt kell fektetni az olyan gyakorlatokra, mint: test-first megközelítés (K1), shift left (K2), retrospektívek (K2)
  - Új szakasz a tesztelésről a DevOps kontextusában (K2)
  - Az integrációs tesztelési szint két külön tesztszintre oszlik: komponensintegrációs tesztelés és rendszerintegrációs tesztelés
- Főbb változások a 3. fejezetben (Statikus tesztelés)
  - A felülvizsgálati technikákról szóló rész a K3 TC-al (felülvizsgálati technika alkalmazása) eltávolítva

- Főbb változások a 4. fejezetben (Tesztelemzés és műszaki teszttervezés)
  - A használatieset-alapú tesztelés eltávolítva (de továbbra is jelen van az Advanced Test Analyst tantervben)
  - Nagyobb hangsúly a tesztelés együttműködésen alapuló megközelítésére: új K3 TC az ATDD használatáról tesztesetek származtatására és két új K2 TC a felhasználói történetekről és elfogadási feltételekről
  - A döntési tesztelést és lefedettséget felváltotta az elágazás tesztelés és lefedettség (egyrészt a gyakorlatban gyakrabban használatos az elágazás lefedettség; másrészt a különböző szabványok eltérően határozzák meg a döntést, szemben az „elágazással"; harmadrészt ez egy apró, de súlyos hiányosságot old meg a régi FL2018 tantervben, amely azt állítja, hogy „a 100%-os döntési lefedettség 100%-os utasításlefedettséget jelent" – ez a mondat nem igaz a döntés nélküli programok esetében)
  - A fehérdoboz tesztelés értékéről szóló rész javításra került
- Főbb változások az 5. fejezetben (A teszttevékenységek menedzselése)
  - A tesztstratégiákra/megközelítésekre vonatkozó rész eltávolítva
  - Új K3 TC a tesztráfordítás becslésére szolgáló becslési technikákról
  - Nagyobb hangsúlyt fektet az Agile-hoz kapcsolódó, jól ismert koncepciókra és eszközökre a tesztmenedzsmentben: iteráció- és kiadástervezés (K1), tesztpiramis (K1) és tesztelési kvadránsok (K2)
  - A kockázatkezelésről szóló rész jobban strukturált négy fő tevékenység leírásával: kockázatazonosítás, kockázatértékelés, kockázatmérséklés és kockázatfelügyelet
- Főbb változások a 6. fejezetben (Teszteszközök)
  - Egyes tesztautomatizálási problémákkal kapcsolatos tartalom csökkentésre került, mivel az alapszintet túlhaladó volt – az eszközök kiválasztásáról, a kísérleti projektek végrehajtásáról és az eszközök szervezetbe való bevezetéséről szóló rész eltávolítva

A magyar nyelvű kiadás az eredeti angol nyelvű tanterv magyar nyelvű fordítása a hivatalosan elfogadott magyar kifejezésgyűjtemény (ISTQB® Glosszárium) alapján.

11. Tárgymutató