Tesztelés gyártásban fejlesztésben

Kovács János

A kiadvány megjelenését a thyssenkrupp Components Technology Hungary Kft. támogatta

thyssenkrupp

Kovács János

Tesztelés – gyártásban – fejlesztésben

első kiadás

A kiadvány megjelenését a thyssenkrupp Components Technology Hungary Kft.

támogatta

Budapest

2018

Köszönetnyilvánítás

E jegyzet megszületéséért hálás köszönettel tartozom:

- a lektorálásért és a konstruktív kritikáért:
  - Fliszár Dániel tesztprojekt vezetőnek (thyssenkrupp Components Technology Hungary Kft.)
  - Illés Dániel tesztmérnöknek (thyssenkrupp Components Technology Hungary Kft.)
  - feleségemnek, Bakos Andrea Katalinnak
- az előadáshoz biztosított katedráért dr. Kovács Balázs egyetemi docensnek (Óbudai Egyetem Kandó Kálmán Villamosmérnöki Kar – Mikroelektronika és Technológia Intézet)
- a megjelenés anyagi támogatásáért a thyssenkrupp Components Technology Hungary Kft.-nek

Tartalomjegyzék

Elöljáróban

Az előadás célja a tesztelés megismertetése a kollégákkal annyi szemszögből, amennyi csak hat másfél órás előadásba belefér, hiszen az ipari tapasztalat az, hogy az egyetemről kikerülő, frissen végzett mérnökök többsége nem, vagy csak felületesen ismeri ezt a területet.

A tesztelés feladata

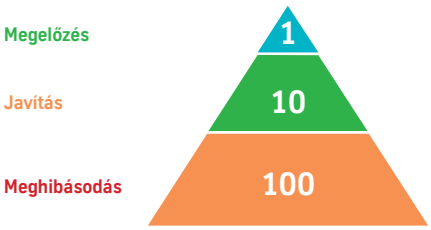
Senki nem szereti, ha az általa megvásárolt termék nem felel meg az elvárásainak, ez ugyanúgy igaz egy elemlámpára, mint egy űrhajóra. Az ilyen meg-nem-felelések (a továbbiakban, hibák) kivétel nélkül anyagi és erkölcsi károkat okoznak a termék előállítási folyamatának minden résztvevőjének, kezdve a forgalmazóval egészen addig a pontig, ahol a hiba keletkezett (és itt nem feltétlenül egyéni felelősségéről beszélünk, hiszen egy szisztematikus hiba forrása ugyanúgy lehet egy helytelenül karbantartott gép, vagy egy nem megfelelően átgondolt folyamat).

Anyagi az okozott kár, ha a kokrét, pénzben kifejezhető költségekről beszélünk. Ez lehet – a teljesség igénye nélkül – szervizköltség, garanciális csere, visszahívásból adódó veszteség (utóbbit tovább bontva: hibaanalízis költsége, anyag- és gyártási veszteség, kötbér, stb.), kártérítések (ha egy hibáról megállapítható, hogy tulajdonosának további károkat – pl. hibás érintésvédelem esetén áramütés – okozott, a gyártó felelősséggel tartozhat azok megtérítésére is, a bíróság döntésétől függően).

Erkölcsi az okozott kár, ha a hiba, vagy az abból származó kár a gyártó és/vagy a forgalmazó rossz hírét kelti, csökkentve a felé mutatott vásárlói bizalmat (természetesen – visszavetve a megrendelések számát – közvetve ez is anyagi kárhoz vezet).

Az ipar történelmében már többször előfordult, hogy akár egyetlen hiba, akár hibák láncolata által okozott anyagi és erkölcsi károk együttesen egy ipari résztvevő csődjéhez vezettek. Az autóipari alkatrészeket gyártó japán Takata 2013-ban a cég hibás légzsákjai miatt kirobbant botránya (csak a Hondákban 100-nál több sérülést és 13 halálesetet okozott a cég nem megfelelő gyártástechnológiája[1]) 2015-ben a cég amerikai csődjét és Japánban történő csődvédelembe vételét okozta. De a tragédiához nem kell, hogy rendszerszintű legyen egy hiba, elég csak a repülőgépekre gondolni, ahol egy nem megfelelő hegesztés is emberek százainak a halálához vezethet.

Ebből látszik, hogy az ilyen hibák kivédése az ipar minden területén megfelelő súllyal rendelkező feladat kell, hogy legyen. Ebből a feladatból vállal oroszlánrészt a tesztelés.



A hibák okozta károktól – a hiba jellegétől függetlenül – el lehet mondani, hogy költsége nagyban függ attól, életciklusának mely pontján sikerült azonosítani a hibát. Ezt az 1/10/100-as szabályként[2] ismerjük. Lényege, ha egy hiba, ha elhárítjuk (prevention/megelőzés), mielőtt bekövetkezne, 1 dollárba kerül, a hibás termék javítása (correction) annak legyártása után, de még a termelésben észlelve már 10 dollárba fog, míg a polcokra kerülve, garanciálisan javítva 100 dollárnyi kárt fog okozni (meghibásodás/failure).

[1] *Tabuchi, Hiroko; Jensen, Christopher. "Now the Air Bags Are Faulty, Too". The New York Times. Retrieved June 25, 2014.*
[2] *"Philip Crosby, 75, Developer Of the Zero-Defects Concept". The New York Times. New York. 2001-08-22. ISSN 0362-4331.*

Természetesen a valóság ettől néhány százalékban eltérhet, de a nagyságrend világos. A tesztelés az életciklus mindhárom fázisában részt vállal.

Egy hibát megelőzni többféleképpen lehet: észlelhetjük, ha egy gép működése nem megfelelő (pl. túlságosan igénybe veszi mechanikusan a munkadarabot, amiből később villamos kontakthibák származhatnak), vagy egy folyamat felépítése lehetne átgondoltabb (pl. a túl sok felesleges mozgatás során sérülnek az alkatrészek), de a dolgozók megfelelő betanítása is a megelőzés része. A teszteredmények itt elsődleges iránymutatásul szolgálnak arra, hogy a jelenlegi folyamatokra nézve azonosítsuk azokat a helyeket, ami után látványosan megnőtt a hibák száma. Az ily módon definiált munkafázisok esetében az erre szakosodott szakemberek javaslatokat tehetnek a gyakrabban előforduló hibák kivédésére, amiket megfelelően alkalmazva a meghibásodások száma jelentősen csökkenthető.

Vitathatatlan, hogy a tesztelők feladatainak oroszlánrészét az teszi ki, hogy a javítás során meggátolható legyen a hibás termék polcokra kerülése. Az összes értékteremtő folyamat munkafázisaihoz hozzárendelődik valamilyen (a terület támasztotta követelményeknek megfelelő mélységű) tesztelési tevékenység, ahol megerősítést nyer annak sikeressége, vagy kudarca az adott termék esetében. Ha valami megbukik az ellenőrzésen, az vagy selejt lesz, vagy (ha a hiba jellege és a vállalt ipari szabvány lehetővé teszi) javítják.

Felmerülhet a kérdés, hogy megfelelően kivitelezett teszt-tevékenységgel hogyan kerülhet ki az értékteremtő folyamatból olyan termék, ami hibát hordoz magában. Fontos hangsúlyozni, hogy kimerítő tesztelés[3] nem létezik, a gyártás/fejlesztés során elvégzendő teszteseteket[4] a termék használata során bekövetkező meghibásodások azonosítását, majd kockázat-alapú súlyozását követően határozzák meg. Nagy bonyolultságú rendszerek esetében tökéletes kockázat-becslést megalkotni nagyon nehéz, ennek köszönhetően elképzelhető, hogy az analízist végzők előtt észrevétlen marad egy-egy rizikófaktor. Amennyiben így történik, és a felhasználóktól visszajönnek a hibabejelentések, rendszer-szintű hiba esetén a tesztelők segítenek azonosítani a annak gyökér-okát[5], hogy (visszatérve a hiba életciklusának elejéhez) megelőzzék a jövőbeli felbukkanását.

Következésképpen elmondható, hogy azért tesztelünk, hogy az általunk előállított termék minőségébe vetett bizalmat növeljük. Ily módon elkerüljük a meghibásodott termékek okozta (közvetett és közvetlen) költségeket, biztonságosabbá tesszük az ember által előállított eszközök használatát, valamint maximalizáljuk az esélyét, hogy a felhasználó elégedett legyen azzal, amiért pénzt adott.

---

[3]Kimerítő tesztelés (exhaustive testing): az elméletben lehetséges legmélyebb szintű tesztelés, amikor a lehetséges felhasználás összes módját, a lehetséges bemeneti értékek összes kombinációját, (szoftver esetében) az összes sort, (hardver esetében) az összes alkatrészt tesztesetekkel lefedik.

[4]Teszteset (test case): Bemeneti értékek és végrehajtási előfeltételek, várt eredmények és végrehajtási utófeltételek halmaza. (Micskei Zoltán, Majzik István: A szofver tesztelés alapjai, https://inf.mit.bme.hu/sites/default/files/materials/category/kateg%C3%B3ria/oktat%C3%A1s/msc-t%C3%A1rgyak/szoftverellen%C5%91rz%C3%A9si-technik%C3%A1k/12/SZET-2012_EA06_teszteles_alapjai.pdf)

[5]Gyökér ok (root cause): Egy probléma valódi oka, amely orvoslásával biztosan megszüntetjük a problémát, annak újbóli előfordulását, továbbá elejét vehetjük sok más potenciálisan kialakuló problémának. (leanszotar.hu)

Az ipari folyamatok

„A folyamatok termelik meg azokat az eredményeket, amelyeket a vállalat eljuttat a fogyasztókhoz.[6]" Mivel a tesztelés minden elemében szorosan kötődik ezekhez az eredményekhez, így lehetetlen róla kontextusán, az ipari folyamaton kívül érdemben beszélni, módszereit, céljait, alapelveit megérteni.

Értékteremtő- és értéket nem teremtő tevékenységek

A folyamatok egyes értelmezések szerint két részre oszthatóak, értékteremtő, és értéket nem teremtő tevékenységekre (amiket veszteségnek is nevezünk). Értékteremtőnek azt a folyamat-részt nevezzük, ahol az alábbi három feltétel mindegyike teljesül:

- a munkadarab megváltozik, készültségi foka növekszik tőle
- ez az a tevékenység, amiért a vásárló/fogyasztó tulajdonképpen fizet
- elsőre jól végezzük el

Mi lehet tehát értékteremtő tevékenység? Összefoglalva: mindent, amit a munkadarabbal (legyen az acéltömb, vagy szoftverkód) végzünk, közben akár hozzáadva, akár elvéve belőle. A végtelen példa közül kiragadva a forgácsolás, a hullámforrasztás, a kód-generálás, az alkatrészek összeszerelése stb.

Az általánosan elfogadott elmélet szerint minden más tevékenység értéket nem teremt, tulajdonképpen veszteség, hiszen önmagában közvetlenül nem növeli a projekt nyereségét (sőt, ára is van, de ez van, hogy megtérül). Ide tartozik az összes olyan munkaszakasz, ahol nem közvetlenül a termékkel dolgozunk, a szállítás, a beszerzés, a logisztika, stb. Ide tartoznak azonban a folyamatba be nem tervezett tevékenységek is, pl. a selejt-előállítás, vagy a felesleges termék-mozgatás.

Az értéket nem teremtő tevékenységeket további két csoportra oszthatjuk: szükségtelen, és szükséges veszteségekre.

Szükségtelen veszteség minden, aminek elhagyása nem befolyásolná hátrányosan a vállalat rövid- és hosszútávú céljainak elérését. Ilyen a gyártásban a fent említett két dolog, a selejtgyártás, ahol nem nehéz belátni e kategória jogosságát, vagy a munkadarabok felesleges mozgatása. Mindkettő kivédhető megfelelően átgondolt folyamatokkal. Szükségtelen veszteség szoftverfejlesztés területén pl. a félreérthető követelmények megalkotása, amikből megfelelő felülvizsgálat hiányában a követelmények nem teljesítése adódhat, de ilyen a nem megfelelő erőforrás-felhasználásból adódó deadlockok[7] létrehozása is. Hogy a tesztelést is említsük, szükségtelen veszteség minden tesztelési tevékenység, amit egy olyan rendszer validációjára fordítottunk, amit végül egy, a kiszállítást blokkoló hiba miatt nem lehet kiadni. A cél az ilyen jellegű veszteségek esetében egyértelműen a teljes megszüntetés.

A veszteségek másik csoportja egyértelműen szükséges, elhagyása nem csak, hogy komolyan megnehezítené a vállalati célok elérését, de el is lehetetlenítené annak működését. Ezzel együtt nem teljesül rá a fent ismertetett három feltétel. Ehhez a csoporthoz tartozik többek között a logisztika, a könyvelés, a beszerzés, a minőségfejlesztés, a HR, az IT (infrastrukturális értelemben legalábbis), a géppark műszaki karbantartása, az adminisztráció és a tesztelés is. A cél ezeknél a tevékenyégeknél ezek minimálisra csökkentése, azaz, hogy mindig

[6] *Hammer, M. – Champy, J. (2001). Reengineering in the Corporation: A Manifesto for Business Revolution. New York*
[7] *Deadlock (holtpont): olyan állapot, ahol több folyamat versenyhelyzetbe kerül egy adott erőforrás használatáért, és emiatt kölcsönösen blokkolják egymást.*

annyi legyen a rájuk fordított erőforrás, amennyit a termék felhasználási területe és/vagy a szerződésben vállalt ipari szabványok még éppen megkövetelnek.

A vállalati folyamatokat nagyban meghatározzák a következő tényezők:

- A vállalat mérete: Egy startup, vagy egy egyszemélyes magánvállalkozás folyamatait tekintve sokkal szabadabb, mint egy multinacionális cég. Ez jórészt abból adódik, hogy kevés ember könnyebben tudja követni a külső szemlélő által első pillantásra kaotikusnak tűnő gyakorlatokat, míg egy nagyvállalat esetében az egyéni kompetencia súlyát felváltja a kollektív tudás, amit a jól dokumentált folyamatok tartalmaznak, és tesznek hozzáférhetővé az új belépők számára is.

- A termék jellege: Általánosságban elmondható, hogy minél egyszerűbb egy adott termék, s leginkább, minél kisebb kockázatokat rejt a használata közben bekövetkezett meghibásodás, annál kevésbé szükséges szigorúan betartott folyamatok alkalmazása, míg az előbb említett tényezők növekedése megnövekedett folyamat-beli fegyelmet követel meg.

- A termék felhasználási területe: Előfordulhat, hogy két termék jellegét tekintve teljesen, vagy közel azonos, ám mégsem ugyanaz a megbízhatóságukkal – ezzel áttételesen az előállításuk folyamataival – szemben támasztott elvárás. Ilyen a konyhakés és az orvosi szike esete, ahol a tervezett felhasználásuk különbségének meg kell mutatkoznia az előállításuk során tanusított tervezési- és gyártási fegyelemben.

- A vállalt/kötelező szabványok: Részben a termék jellegéből és felhasználási területéből, részben az értékesítési terület törvényeiből, részben a vállalat által sugallni kívánt minőségi elköteleződésből adódóan az ipari folyamatok meghatározásában segítenek/köteleznek bizonyos szabványok, melyek olyan különböző célokat hivatottak garantálni, mint a termék megbízhatósági-, biztonsági foka, ökológiai lábnyoma, stb.

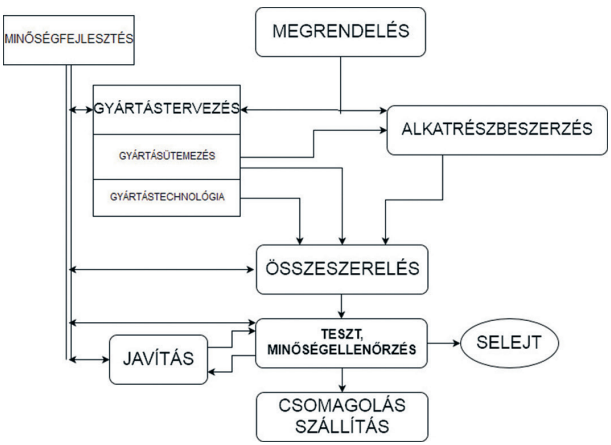
- Az életciklus-modell (fejlesztésnél): A termék jellege tapasztalati úton meghatározza azt is, hogyan érdemes fejleszteni azt, hogy a ráfordított erőforrás lehető legnagyobb százalékban hasznosuljon. Sokkal kevesebb fegyelmet igényel a vízesés-modell, vagy a prototyping, mint az agilis életciklusú fejlesztések, vagy a V-modell. Az életciklus modellekről a fejezet későbbi szakaszában esik majd szó.

Az elmúlt évtizedekben megfigyelhető, hogy egyre nagyobb figyelmet kapnak a vállalati folyamatok és fejlesztésük, ugyanis általuk minimalizálható a cég működése során keletkező veszteség, kevesebb a váratlan helyzet, átláthatóbb a működés, és stabilabb a stratégiai értelemben vett jövőkép.

Gyártás – megrendeléstől a csomagolásig

Mivel az ipari forradalom óta a termelés meghaladta a néhány fős műhelyek, manufaktúrák világát, ahol kis méretekben – tulajdonképpen az egyszeri szemlélő számára is viszonylag könnyen átláthatóan – folyt a munka, bármit gyártani jól definiált munkafolyamatokat igényel.

Elengedhetetlen, hogy a termelés minden résztvevője tudja, mi a dolga, mettől meddig tart a felelősségi köre, és az általa végzett munkafolyamatoknak mi a bemenete, a kimenete, illetve milyen külső résztvevőkkel áll kapcsolatban. Ez nagyon változatos ismeretigényű beosztásokat eredményez: egy összeszerelésben dolgozó operátornak elég ismernie a saját feladatát, egy karbantartó mérnöknek az általa felügyelt gépek működését, felépítését és a gyártásban betöltött szerepét, míg egy folyamatfejlesztőnek tisztában kell lennie a sor minden állomásával, azok feladataival, a belőlük fakadó lehetséges buktatókkal, stb. Amennyiben ez a tudás a helyén van, a gyár maga is úgy üzemel, mint egy jól beállított gép.



A 0. lépés egy termék előállításában mindig a megrendelés érkezése. Ez történhet „házon" belülről, és kívülről is. Előbbi eset akkor következik be, ha a vállalat saját fejlesztésű termékéről van szó, míg utóbbinál bérgyártásról beszélünk egy külső[8] ipari résztvevő számára. A gyártásnak küldött megrendelésnek a következőket kell tartalmaznia:

- darabszám
- határidő (jellemzően nem egy ütemben történik a gyártás, hanem a megrendelőnek több lépcsőben van szüksége a termékre, hogy folyamatos legyen az áruellátása, és kevesebb raktárkapacitását kösse le)
- villamossági/elektronikai tervek: NYÁK terv, vezetékelési rajz, stb.
- mechanikai tervek: fúrási információkat tartalmazó gerber file-ok, marási utasítások
- BOM – bill of materials – egy, a termékhez szükséges teljes anyag- és alkatrészigényt magába foglaló lista

[8] *Third party*

A megrendelés beérkezte után a gyártástervezés következik, aminek két dimenziója van: egy logisztikai/gazdasági, a gyártásütemezés, illetve egy műszaki, a gyártástechnológia.

A gyártásütemezés az erőforrásoptimalizácó világa. Célja annak megvalósítása, hogy a vállalat úgy tarthassa sikeresen a megrendelő által szabott határidőket, hogy a lehető legkevesebb erőforrást vonja be, ezzel maximalizálva a befektetett energia és a hozam arányát. Mivel a legkisebb gyárakban is általában egyszerre több terméket gyártanak, több projektet szolgálnak ki, így könnyen adódhat konkurencia helyzet az erőforrások felhasználásánál, mert gyakran van szüksége ugyanarra a gépre, sorra, szakemberre egyszerre több projektnek is. Aki ezen a területen dolgozik, erős tárgyalási képességekkel kell rendelkezzen, a döntés során pedig, hogy ki is „nyerte meg" a szóban forgó erőforrásokat, részben ez, részben pedig az egyes projektek, megrendelések súlya, prioritása dönt. A gyártásütemezés ily módon ún. időablakokat, gépidőket szerez magának. Ezen gépidők alatt kell majd legyártani, összeszerelni a megrendelés teljesítéséhez szükséges, megfelelő minőségű terméket.

A gyártástervezés másik fele a gyártástechnológia, ami a technikai hogyan-ra keresi a választ. A megrendelést követően a technológus szakemberek (gyakran a fejlesztőkkel együttműködve) meghatározzák a legalkalmasabb módszert a termékek előállítására, ami (hasonlóan a gyártástervezéshez) a lehető legkevesebb anyag- és energiaráfordítással eredményezi a minőségi követelményeit teljesítő termék előállítását. A gyártástechnológiai feladatok legnagyobb része a tényleges gyártás elindulása előtt történik: ekkor tervezik meg a tulajdonképpeni gyártósort, annak egyes munkaállomásait, azt, hogy ezeknél mik történjenek a munkadarabbal, röviden, hogy milyen lépésekben jussanak el az üres alkatrésztől vagy nyersanyagtól a kész termékig. Az így megtervezett technológiai lépéseket egy próbagyártás során ellenőrzik, ami a kimeneti eredményeitől függően lehetőséget ad az esetleges hibás elképzelések kijavítására anélkül, hogy a tömeggyártás költségeivel járna.

Ha a gyártást megtervezték, még nem ért véget sem a gyártásütemezés, sem a gyártástechnológia feladata. Mivel az üzem nem egy statikus képződmény, a körülmények folyamatosan változnak: különböző meghibásodások miatt leállnak gépek, megbetegszenek, szabadságra mennek a dolgozók, új, még sürgősebb megrendelések esnek be, egy adott technológiai lépés nem hozza a várt hatékonyságot, valamilyen okból kifolyólag megugrik a selejtszám, stb. Emiatt folyamatos utánállítgatásokat igényel mind az ütemezés, mind a technológia, mert az ilyen típusú rugalmasság nélkül a határidők és a szükségtelen veszteségek megcélzott szintje tarthatatlanok lesznek.

A gyártásütemezés határozza meg a megrendelés teljesítéséhez szükséges alkatrészek mennyiségét is. Amint elkészült a gyártási ütemterv (meghatározva, milyen időablakokban mekkora példányszámot szükséges előállítani az adott termékből), az alkatrészbeszerzésen a sor, hogy a BOM-ban szereplő minden hozzávaló rendelkezésre álljon, amikor szükség lesz rá. Ez történhet közvetítő kereskedőkön keresztül (főleg kisebb darabszámok esetén van ennek realitása), vagy közvetlenül az alkatrészgyártókkal együttműködve. Mivel az ipari nyersanyagok világpiaci elérhetősége számos tényező függvénye, célszerű minél előbb elindítani ezt a folyamatot, mert vannak alkatrészek, amikre nagy darabszám[9] esetén igen nagy, egy évnél is hosszabb várakozásra lehet számítani. A XX. század közepéig az ún. Just-In-Case (JIC) szemlélet jellemezte az alkatrészbeszerzést: mindenből rendeltek, amennyit a költségvetés elbírt, hogy egy nagyobb hiány ne akadályozza a termelést, jellemző volt az óriási készletek felhalmozása.

---

[9] *Gyakran használt elektronikai alkatrészek esetén milliárdos megrendelési tételekről beszélünk!*

A Toyotánál az 1950-es években rájöttek[10], hogy megfelelően átgondolt alkatrészgazdálkodással sokat csökkenthetnek a raktározással, logisztikával kapcsolatos költségeiken, amennyiben mindenből csak annyit tartanak, amennyire az üzemben épp szükség van. Ez az úgynevezett Just-In-Time (JIT) szemlélet, amit azóta a legtöbb ipari szereplő szektoron belül és kívül egyaránt átvett, és alkalmaz. Emiatt egy adott alkatrészből úgy adják le a rendelést, hogy mindig csak annyi érkezzen egyszerre, amennyit az adott gyártási ütem megkíván, belekalkulálva a selejtekből adódó többletigényt.

Amint a próbagyártás megfelelő eredménnyel szolgál, valamint adott a szükséges emberi, infrastrukturális és tárgyi erőforrás is, megkezdődik a termék megmunkálása, összeszerelése. Itt nehéz bármi konkrétumot írni, hiszen a technológiai folyamatok a termék jellegének függvényei, értelemszerűen más kell egy elektronikai alkatrész előállításához, mint egy lengéscsillapítóhoz, vagy egy notebookhoz.

A gyártási, összeszerelési folyamatot a minőségellenőrzés követi, ami jelen esetben egyet jelent a teszteléssel[11]. Az, hogy konkrétan mi történik itt, szintén a gyártott termék jellegéből következik, és széles skálán változik az egyszerű szemrevételezéstől a nagy értékű ipari mérőautomatikával végzett funkcionális tesztekig. A tesztelés célja minden esetben annak megerősítése, hogy a termék megfelel a megrendelő elvárásainak. Szempontjai a következők lehetnek:

1. Összeszerelési, megmunkálási minőség: elképzelhető, hogy a termék jól láthatóan nem felel meg a vevő céljainak, mert a gyártási folyamat során hiányos marad, vagy sérül.
   Az ilyen jellegű hibák kiszűrése az alacsony összetettségű termékek esetében a legegyszerűbb, kompetencia- és műszerigénye ennek a legalacsonyabb. Jellemző, hogy a tesztelők külön szervezeti egysége esetén nem is hozzájuk tartozik, hanem a minőségellenőrzéshez, vagy magához a gyártáshoz, a feladat ellátása pedig ritkán igényel betanított kollégáknál magasabb képzettségű munkaerőt.
   Nagy bonyolultságú termékek esetén az ilyen hibák kiszűrése magas fokú műszerezettséget és automatizáltságot igényel, az elektronikai gyártásban ún. In-Circuit-Testing módszerrel (lásd később) keresik meg az esetlegesen hiányzó, vagy nem megfelelő alkatrészeket.
2. Működés robosztussága: minden termék esetében tudni lehet, hogy a vevő milyen alkalmazásra szánja azokat, felhasználásuk során mekkora terhelésnek lesznek kitéve. A megrendelő ezeket a terheléseket a követelményekben specifikálja, többségében külön kezelve az elviselendő határértékeket folyamatos, valamint meghatározott idejű terhelés esetén. Ilyen terhelés lehet egy dugattyú hajtókarjára ható nyomaték, vagy egy motorvezérlő áramkör kapcsolótranzisztorán átfolyó áramerősség is.
3. Funkcionalitás: magától értetődő igény, hogy a termék azt csinálja, amit a vevő kért. Az ilyen szempont szerint elvégzett tesztek során is a vevői követelmények adják a tesztesetek alapját, ahol a bemeneti jelek különböző kombinációira vizsgálják a kimeneti jeleket. Ilyen lehet egy tévé végtesztelése is, különböző színskálák betöltésével és a képernyőn megjelenő kép egy mintaképpel történő összevetésével, vagy egy beágyazott áramkör bemeneteinek szimulált jelekkel történő stimulálása, miközben a kimeneten megjelenő jeleket összevetik egy referenciaérték-maszkkal.

---

[10] *The Productivity Press Developement Team: Just-in-Time. Kaizen Pro Kft. Budapest, 2011*

[11] *A tesztelés a minőségellenőrzéshez tartozik, ugyanakkor – különösen ott, ahol a vállalat nagysága megköveteli – gyakran külön szervezeti egységként funkcionál, az itt alkalmazott technika és az abból felmerülő feladatok speciális mivoltából adódóan. A minőségellenőrzés azonban nem kizárólag a tesztelésből áll, hozzá tartoznak pl. a különböző külső- és belső auditok is, amin keresztül biztosítják az egyes folyamatok során betartott gyártási fegyelmet, ami összefüggésben áll a termék minőségével.*

4. Megbízhatóság: fontos, és érthető szempont a megrendelő részéről, hogy a kész termék a garanciaidőt meghibásodás nélkül teljesítse. Ezt túlnyomórészt a tervezés során teljesítik, a felhasznált anyagok minőségének kiválasztásával, ám – mivel ezek (költségoptimalizációs céllal) többnyire úgy lesznek meghatározva, hogy éppen, hogy kibírják a vállalt garanciaidőt – elengedhetetlen a megfelelő gyártási fegyelem. Mivel a gyártásból kikerülő termékek meghibásodási rátája az ún. kádgörbével írható le, ahol láthatóan a bejáratás és az elhasználódás fázisaiban a legmagasabb a hiba felbukkanásának valószínűsége, az ilyen irányú tesztelés célja a termék bejáratás utáni állapotúra öregítése. Ezt különböző praktikákkal (pl. magas hőmérsékleten üzemeltetés), vagy nyers erővel érik el (pl. meghatározott óráig/napig járatják a terméket műterheléssel).

Természetesen a gyakorlatban (a termék összetettségétől függően) nem egyszer fordul elő a gyártási folyamat során az összeszerelés és a tesztelés, hanem többször is követhetik egymást.

Mivel a tesztelés feladata a hibás termékek kiszűrése, ugyanakkor a teszt maga nem ad közvetlenül értéket a termékhez, a minőségellenőrzési állomások, tesztautomaták elhelyezésénél is törekedni kell az optimalizációra. Az ilyenkor a gyártástervezésben alkalmazott vezérelv szerint azokon a helyeken, ahol jelentős a munka elrontásának esélye, vagy akkor, ha a legutóbbi ellenőrzést követően végzett munkák elrontásának valószínűsége összeadva meghalad egy bizonyos értéket, minőségellenőrzést kell végezni.

Ha egy termék valamilyen okból megbukik a minőségellenőrzésen, két lehetősége van: amennyiben a minőségellenőrzést végző dolgozó javíthatónak ítéli, az erre szakosodott technikus kézzel végzi el a szükséges korrekciókat, majd visszakerül a tesztelésre, hogy megerősítést nyerjen a javítás sikeressége. Gyakran azonban a javítás nem lehetséges annak nehézsége, vagy gazdaságtalansága miatt. Ez esetben selejt lesz a termékből, amit gyakran az erre szakosodott, ipari hulladék kezelő cégek szállítanak el, és használnak fel annyi mindent belőlük, amennyit csak lehetséges, amit pedig nem, attól (ideális esetben) környezetbarát módon megszabadulnak.

A tesztek során jónak ítélt termékek a csomagolókhoz kerülnek, ahol dobozolva, raklapokra pakolva gondoskodnak a megrendelőhöz történő elszállításukról.

Fontos résztvevő még a gyártásban a minőségfejlesztés is, akikről korábban is esett már szó, a folya-

matok hatékonyságának növelésénél. Mivel az ő dolguk a veszteségek minimalizálása, ezért a gyártásban résztvevő összes résztvevővel kapcsolatban állnak, igyekezvén kidolgozni a lehetséges fejlődési utakat. A tesztelésből származó statisztikák kulcsfontosságúak a számukra, hiszen általuk kimutathatók a gócpontok a selejtszámok szempontjából, a hiba keletkezésénél kockázatos technológiai lépések, vagy a kompetencia hiánya. Az ezek kiderülését követő analízis segítségével a minőségfejlesztési szakemberek javaslatokkal élhetnek a gyártásban résztvevő dolgozók felé, amiket, ha bizonyítást nyert igazuk, hivatalos munkautasítás szintjére emelve bevett gyakorlatokká tesznek.

A gyártási folyamatok során végzett teszt tehát, amellett, hogy biztonságosabbá teszi a kikerülő termékek használatát, és bosszúságot spórol meg a végfelhasználóknak, a költségének tízszeresére rúgó veszteségektől kíméli meg a gyártó vállalatot a selejt kiszűrésével, meggátolva, hogy egy nem működő termékbe ölje az anyagot és munkát a cég, valamint hogy garanciális költségek merüljenek fel a sorról kikerülő hibás termékek miatt.

Tesztelés az elektronikai gyártásban

Mint az előző fejezetben olvasható, a tesztelésnek a gyártásban két feladata van: meggátolni, hogy nem megfelelő termék kerüljön a kereskedelembe, vagy a végfelhasználóhoz, így megelőzve a garanciális követelésekből eredő költségeket, valamint elejét venni, hogy egy olyan munkadarabba öljön a vállalat további energiát, ami már a vizsgálat idején sem teljesíti a felé támasztott követeléseket[12].

Azt, hogy mikor kell ellenőrizni a munkadarab minőségét, az határozza meg, mekkora az esélye annak, hogy a megelőző munkaállomás, vagy munkaállomások során hibát követtek el. Ezt empirikus úton szokták eldönteni a gyártás tervezési fázisában.

A másik fontos kérdés ugyanekkor, hogy az adott ellenőrzési feladatot manuálisan végezzék el, vagy automatizáltan.

Manuális teszt, minőségellenőrzés

A minőségellenőrzés legnagyobb múltra visszatekintő módszere az, amikor emberi kéz által, emberi ítélőképességre bízva dől el, hogy a munkadarab minősége megfelel-e a vevői követelményeknek. Ide tartozik a nyomtatott áramköri ábra nagyító alatt, szemrevételezéssel történő vizsgálata épp úgy, mint a frissen beültetett alkatrészek ellenőrzése kamerák képén, de ide sorolható az is, amikor a gyártósorról kikerülő kész tévén erre a célra készített képeket, képsorozatokat jelenítenek meg, hogy ez alapján a minőségellenőr megállapíthassa a kijelző esetleges hibáit.

Az, hogy mikor lehetséges, mely esetekben éri meg manuális tesztet alkalmazni, a következő tényezők függvénye:

- Amennyiben a termék bonyolultsága, valamint az alkalmazási terület (az ezzel vállalt szabványokkal egyetemben) megengedi a minőség emberi munkaerő által történő megítélését. Míg egy tömegkereskedelembe kerülő kvarcóra kijelzője ellenőrizhető manuálisan, egy, a repőlőgépiparban felhasználni tervezett hasonló kijelző már körültekintőbb tesztelést kíván, csak úgy, mint egy összetettebb termék, pl. egy hűtőgép vezérlője. Ez tekinthető az elsődleges irányelvnek.

[12] *A folyamatelmélet területén az aktuális munkaállomást követő gyártási fázis is vevőként értelmezendő.*

- Ha a gyár olyan országban működik, ahol az emberi erőforrás fajlagos költsége olcsóbb, mint az ugyanezt a munkát végző tesztautomatáé[13], a manuális minőségellenőrzés az optimális választás.[14]
- Ha egy üzem sokféle terméket gyárt, kis szériában, nem feltétlenül éri meg vállalni az automatizálás költségeit, mert a gépek átállítása, univerzalitásának költségei már nem feltétlenül érik meg az automatizáltsággal elért nyereséget.
- A minőségellenőrzés során elvégzendő feladat összetettsége szintén fontos, hiszen adódnak feladatok, ahol az emberi ítélőképesség szerepe fontos, de legalábbis nagyon költséges lenne azzal egyenértékű tesztautomatát létrehozni.

A fentiekből látható, hogy manuális tesztet akkor alkalmazunk, ha a gyártott termék megengedi, valamint ha gazdaságosabb az emberi munkaerő, mint a gépesítés.
A manuális teszt előnyei:

- Az ember árnyaltabb döntésekre képes.
- Rugalmasabb, adaptív.
- Bizonyos régiókban olcsó a munkaerő.
- Változatos tesztesetek elvégzésére teremt lehetőséget egyetlen munkaállomáson.

A manuális teszt hátrányai:

- Kevésbé reprodukálható.
- Az emberi döntések pontossága nem fogható a műszerekéhez.
- Az emberi természet hiányosságai.

Automatizált tesztelés az elektronikai gyártásban

Amennyiben a fent felsorolt feltételek ill. szükségletek teljesülnek egy termék gyártásánál, az automatizált tesztelés az egyetlen megoldás a munkadarab minőségi besorolásához. Az automatizált tesztelés sajátossága, hogy minden esetben valamilyen – az adott mérési, ellenőrzési feladatra optimalizált – célhardverrel történik, különböző, általában (de nem kizárólagosan) villamos méréssel.

Ezeket a tesztautomatákat oly módon helyezik el a gyártási folyamatban, hogy lehetőség szerint integrálva legyenek a meglévő gyártósorba, elkerülve ezzel a felesleges mozgatást. A tesztautomaták helyét ugyanazzal a módszerrel határozzák meg, mint az összes többi minőségellenőrzési állomást, a megelőző megmunkálások elrontásának kockázat-elemzésével.

Az, hogy mikor lehetséges, mely esetekben éri meg automatizált tesztet alkalmazni, a következő tényezők függvénye:

- Amennyiben a termék bonyolultsága, valamint az alkalmazási terület (az ezzel vállalt szabványokkal egyetemben) nem engedi meg a minőség emberi munkaerő által történő megítélését. Egy biztonságkritikus

---

[13]Ezt a költséget az amortizáció és az üzemeltetés (ami az energiavelvételből és a karbantartásból adódik) költségének összege adja.
[14]Ehhez –közvetett módon magába az üzemeltetési költségbe épülve – hozzá tartozik az adott országban elérhető szakértelem is, hiszen egy (általában) nagy bonyolultságú tesztautomata bár kis létszámú, de magas kompetenciájú munkaerőt igényel. Ha ez nincs jelen valahol, csak igen magas bérköltségekkel lehet oda telepíteni, ami könnyen gazdaságtalanná teheti az automatizált tesztelést.

alkalmazású terméknél (pl. vasúti fékrendszer) nem megengedhető az emberi ítélőképesség alacsonyabb megbízhatósága.

- Ha a gyár olyan országban működik, ahol az emberi erőforrás fajlagos költsége drágább, mint az ugyanezt a munkát végző tesztautomatáé, az automatizált minőségellenőrzés az optimális választás.
- Ha egy üzem termékportfóliója beosztható néhány hasonló jellegű termékre, gazdaságosabb megvalósítani e termékek automatizált tesztjét is, a tesztautomaták különböző gyármányokkal való kompatibilitását biztosító csatlakozóadapterek alkalmazásával.
- A minőségellenőrzés során elvégzendő feladat összetettsége szintén fontos, hiszen adódnak feladatok, ahol az emberi ítélőképesség árnyaltsága nem hordoz igazán üzleti értéket (pl. diszkrét bemeneti- és kimeneti értékeket tartalmazó tesztesetekkel ellenőrzött, digitális rendszerek).

Az automatizált tesztek előnyei:

- Magas szintű megbízhatóság (a mérés és az elvégzett feladat szempontjából egyaránt).
- Reprodukálható mérések.
- Nagyságrendekkel gyorsabb, mint a kézi tesztelés.
- Monoton feladatok elvégzésére is alkalmas.
- Mostoha körülmények (pl. magas hőmérséklet, nagy nyomás, stb.) között is könnyebben elvégezhető minőségellenőrzés.
- Régiótól függően olcsóbb, mint az emberi munkaerő.

Az automatizált tesztek hátrányai:

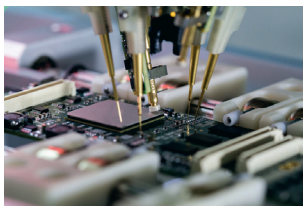
- Üzemeltetése nagy szakértelmet igényel, csak úgy, mint a tesztesetek programozása. Egy tesztautomatánál nem mindig lehet biztosan tudni, hogy egy bukott teszt esetében a termék rossz, vagy a tesztautomata, esetleg a mérési elv maga. Emiatt olyan munkaerő szükséges, aki a méréstechnikában magas fokú tudással rendelkezik, hogy végére tudjon járni az ilyen problémák okainak.
- Árnyalt döntések meghozatalára nem mindig a legalkalmasabb, kevésbé rugalmas, mint az ember.
- Egy gép kevésbé változatos tesztesetek elvégzésére alkalmas, mint egy emberi minőségellenőr.

Tesztautomaták az elektronikai gyártásban

Az automatizált tesztelés különböző szintjeinek bemutatása előtt feltétlenül szükséges az egyes tesztautomata típusok bemutatása, hogy műszakilag kontextusba lehessen helyezni az egyes tesztelési tevékenységeket.

- Villamos mennyiségek mérésére használt tesztautomaták felépítése a tekintetben megegyezik, hogy lelkük egy ipari számítógép, ami az üzemi hálózatról és/vagy egy helyi terminálról kapott utasításokat követve tölti be az aktuálisan futtatandó mérési programot. Az ipari számítógép speciális interfész-buszokon kapcsolatban áll a mérőautomata perifériáival (feszültség- és áramgenerátorok, multiméterek, műterhelések, adatbusz-emulátorok), és ezeket vezérelve, az egyes mérőpontokat a munkadarabra kapcsolva összegyűjti a multiméterek kimenetein megjelenő mért értékeket tesztlépésről tesztlépésre.

o Repülőtűs (flying probe) mérőberendezés



Kép forrása: SPEA

A villamos mennyiségek mérésére használt tesztautomaták leglátványosabb típusa az ún. repülőtűs berendezéseké. Ezek esetében mérőtűkben végződő robotkarok csatlakoznak a tesztprogramban meghatározott pontokra, és mérik meg a közöttük meghatározott villamos mennyiséget.[15]

A repülőtűs mérőautomata előnye, hogy nagyon rugalmas annak tekintetében, hogy mit is mér: átállási ideje egy adott típusú munkadarabról a másikra nagyon rövid, hardveres beállítást nem igényel, elég csak átállítani a futtatandó teszt programját. A mérőpontokat előre beállított koordináták segítségével találja meg (amit sok esetben már kamerás pontosítás is támogat), így nagy szabadságot biztosít a tesztmérnököknek a tesztelhető munkadarabok számának terén.

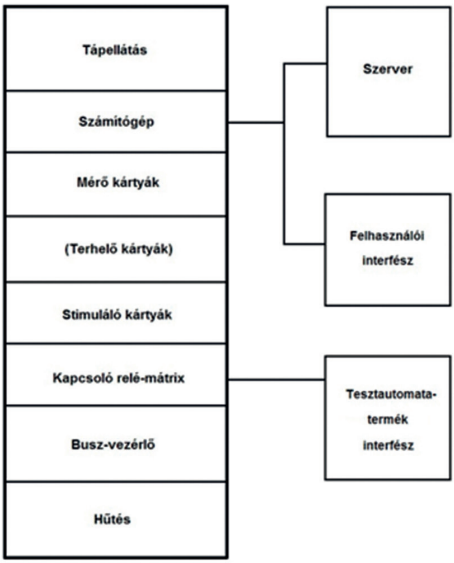
Hátránya mozgó alkatrészek miatti fokozott karbantartásigény, illetve a tűágyas tesztautomatákhoz képesti jóval lassabb mérési sebessége.

Alkalmazása olyan gyártósoroknál indokolt, ahol sok különböző típusú, kis darabszámú sorozat követi egymást, ilyen a prototípusgyártás, vagy az olyan készülékek, berendezések gyártása, ahol a hozzáadott érték nagy, viszont a termék jellegéből adódóan nem folyik tömegtermelés (pl. ipari mérőműszerek előállítása).

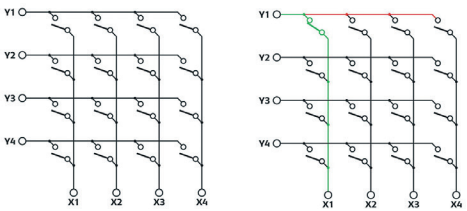
---

[15] *Azért használjuk a semleges „villamos mennyiség" meghatározást, mert a mennyiség jellege (ellenállás, feszültség, kapacitás, induktivitás, áramerősség) tesztről tesztre, sőt, tesztesetről tesztesetre változhat.*

o Tűágyas mérőberendezés

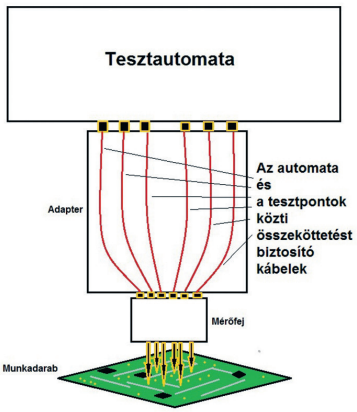


Míg a kis darabszámú sorozatok gyártása esetében megfizetődik a repülőtűs tesztautomaták alkalmazása, a tömeggyártásban szükség van egy hatékonyabb megoldásra. Mivel a nagy darabszámok miatt ritkán (napi, sőt, akár heti néhány alkalommal) van szükség a gépek új típusra való átállítására, a mérőtűk robotkaros mozgatása elhagyható. Ehelyett a tesztelt munkadarab felépítését ismerve rögzített tűket helyezünk el a lehetséges mérőpontok helyén, és egy relémátrix[16] segítségével ezek között a tűk, valamint a tesztautomata perifériái között galvanikus kapcsolatokat teremtünk és szakítunk meg, a tesztprogramnak megfelelően.



[16] *Relémátrix: egy multiplexerhez hasonlóan arra szolgál, hogy digitálisan vezérelve kapcsoljon össze egy kimenetet több lehetséges bemenettel, ily módon lehetővé téve számos villamos összeköttetést minimális vezetékezéssel a pillanatnyi igények szerint.*

A tűágyas mérőautomata előnye, hogy a tesztek során megspóroljuk a tűk robotkaros mozgatását, ami során a repülőtűs méréseknél nem jutunk érdemi adathoz, pusztán azért szükséges, hogy az egyik áramút után a következő áramutat is létrehozzuk. A relémátrix lehetővé teszi, hogy néhány ms-ra (a relék átkapcsolására szükséges időre) rövidítsük ezt, munkadarabonként akár több percet is spórolva a teszten. Ez a repülőtűs berendezésekhez képest óriási kapacitás-növekedéssel jár.
Hátrányuk azonban, hogy típusváltás esetében nem elég a szoftvert kicserélni, hanem a gépen magán is állítani kell. Az, hogy rögzített tűkkel lehessen több típusú gyártmányt is vizsgálni, úgy valósítható meg, hogy speciális adaptereket használnak a munkadarab fajtájától, de legalábbis termékcsaládjától függően. Ilyen adapter felelős a relémátrix és a termék közötti vezetékezésért, valamint a tűk rögzítéséért is.



Ez amellett, hogy értelemszerűen jelentősebb időveszteséggel jár (hiszen amíg az adaptereket cserélik, nincs lehetőség tesztelni), növeli a hibalehetőségek számát is.

- Nem villamos mennyiségek mérésére használt tesztautomaták is használatban vannak az elektronikai gyártásban, ugyanis egy berendezésnek lehetnek nem elektromos, kritikus fontossággal bíró kimenetei is: kijelzők, hangszórók, radarok, fényforrások esetében például nem nehéz belátni, hogy nem nyújt kellő információt pusztán a vezetékezésen, a csatlakozókéseken megjelenő villamos mennyiségek mérése. A legkézenfekvőbb megoldás az emberek által látható, hallható mérendő tulajdonságok esetén a manuális teszt, de van, hogy a gyártás volumene, az ipari szabványok, vagy a vállalat minőségirányítási gyakorlata nem teszi lehetővé ezt. Ilyen esetekben (és persze, ha más minőségű mennyiségek, pl rádió- vagy nagy frekvenciás hanghullámok mére a cél) az ipari méréstechnika speciális tesztautomata perifériákat kínál: kamerákat, mikrofonokat, antennákat, amik éppúgy csatlakoznak egy ipari számítógéphez (nem ritkán robotkarra rögzítve, a munkadarabra minél több szögből történő „rálátás" miatt), mint egy multiméter: kész, referencia-adatokkal összevethető bemeneteket szolgáltatva a teszthez.

Az automatizált tesztelés szintjei az elektronikai gyártásban

In-Circuit Testing (ICT)

A gépesített minőségellenőrzés legalacsonyabb szintje az ún. In-Circuit Testing (a továbbiakban ICT). Ekkor a még minden esetben tokozatlan NYÁK lemez (áramköri elemekkel beépítve vagy anélkül) meghatározott pontjai közötti villamos tulajdonságokat mérik meg, szakadások, rövidzárak, beépítetlen vagy nem megfelelően beépített alkatrészek után kutatva.

Amikor egy munkadarabot ilyen tesztnek vetnek alá, nem foglalkoznak a funkcionalitásával, hanem mindösszesen a strukturális minőségét vizsgálják. A mérések alapja a villamos terv, a tesztesetek fejlesztésénél a NYÁK-on belül különböző áramutakat határoznak meg, a kapcsolási rajz segítségével specifikálva az egyes áramutakon mérendő villamos mennyiségeket.

Az ICT teszt során alkalmazott tesztautomaták tulajdonképpen nagy összetettségű, programozható multiméterek, ahol az egyes tesztpontokra történő kapcsolódást repülőtűk, vagy tűágyas berendezések segítségével érik el.

Funkcionális teszt

Amikor az ICT segítségével megállapították, hogy a munkadarab szerkezetileg rendben van, a következő lépés annak vizsgálata, hogy tudja-e teljesíteni, amit a vevői követelményekben elvárnak tőle. Néhány példa:

- Terhelhetőség: a használatba vételt követően várható terhelés, amit a terméknek el kell viselnie, specifikálva van. A teszt során olyan értékeket kapcsolunk a vizsgált eszköz bemeneteire, aminek következtében eléri ezt a maximális terhelést, és egy (általában a követelményekben szintén meghatározott) ideig ott is tartjuk.
- Működés: minden elektronikus eszköz esetében elmondható, hogy különböző bemenetekre különböző kimenetekkel kell, hogy reagáljon. Ezeket oly módon a legegyszerűbb vizsgálni, hogy az adott bemenetekre a használatban várható tényleges szenzorjelek szimulációját kapcsoljuk, miközben vizsgáljuk, hogy a kimeneten az történik-e, ami ilyen esetekben elvárható.
- Működés tartományai: általában minden terméknél rögzítve van, milyen környezeti tulajdonságok (hőmérséklet, nyomás, páratartalom, stb.) mellett kell használható állapotban maradnia[17]. A teszt során a vizsgált munkadarabot a még kibírandó értékre beállított környezetbe helyezik (pl. klímakamra), majd lefuttatnak rajta egy, az előző ponthoz hasonló, helyes funkcionális működést vizsgáló tesztet.
- Megbízhatóság: Egy termékről elmondható, hogy meghibásodási rátája az ún. kádgörbével írható le. Életciklusa három szakaszra osztható, az elsőben, közvetlenül a gyártás után megmutatkoznak a korai hibák, majd ezek száma folyamatosan csökken. A második szakaszban a meghibásodási valószínűség alacsony értéken stagnál. Ezt követi egy harmadik szakasz, ahol a ráta ismét megnövekedik, az öregedés miatt. Mivel a vállalatok számára érthető cél, hogy a termék csak az első szakaszt köve-

[17] *Ez néha a nem kívánt garanciális megkeresések, perek elkerülése, néha az emberi élet tényleges védelme miatt szükséges.*

tően kerüljön a vevőhöz, különböző technikákkal (magas hőmérsékleten és/vagy terhelésen üzemeltetés néhány óráig) öregítik azokat még a gyártás során. Eközben folyamatosan monitorozzák a működésüket, így elérve, hogy a gyártásból kikerülő késztermék megbízhatósága stabil legyen, tovább csökkentve a garanciális követelésekből származó veszteségeket.

Tesztfejlesztés az elektronikai gyártásban

A manuális és az automatizált tesztelésnél egyaránt fontos, hogy meghatározzuk, mit várunk el az a vizsgált terméktől, adott körülmények mellett.

Ezt emberi minőségellenőrzés esetében egy termékenként és/vagy vizsgálatonként részletes munkautasítást jelent, ami képekkel, ábrákkal adja az ellenőr tudtára, melyik munkadarabon mit kell megvizsgálnia.

Automatizált tesztek esetében a tesztfejlesztés többnyire két részre oszlik:

- Tesztforgatókönyv[18] és tesztesetek[19] megalkotása: Szükséges lépések kidolgozása arra, hogy a termék kipróbálása során érintsük a követelmények egészét vagy azon részét, aminek a vizsgálata megfelelő bizonyítékot szolgáltat arra, hogy a termék a vevői igényeknek megfelelően működik.

Példa:

1. Tápfeszültség 12V-ra állítása.
2. Bemenet1 – Bemenet2 – Bemenet3 beállítása a tesztesetnek megfelelően.

(teszteset1: 0-0-0; teszteset2: 0-1-0; teszteset3: 1-1-0; teszteset4: 1-0-0)

3.1 Kimenet1 értékének ellenőrzése 30ms múlva

(teszteset1: 1; teszteset2: 1; teszteset3: 0; teszteset4: 1)

3.2 Kimenet2 értékváltozásának időmérése, ha a teszteset megkívánja

(teszteset1: <15ms; teszteset2: <15ms; teszteset3: -; teszteset4: -)

4.1 Bemenet3 – Bemenet4 beállítása a tesztesetnek megfelelően.

(teszteset1: 0-0; teszteset2: 0-0; teszteset3: 1-1; teszteset4: 1-1)

4.1 Tápfeszütsé 15V-ra állítása

5. Kimenet1 értékének ellenőrzése 30ms múlva

(teszteset1: 0; teszteset2: 0; teszteset3: 1; teszteset4: 1)

6. Tápfeszültség 0V-ra állítása

- Gyártásbeli tesztfejlesztés: A futattandó tesztek forgatókönyvét és teszteseteit többnyire a termék fejlesztésében dolgozó szakemberek hozzák létre, jól ismerve a követelményeket, amik ellenében dolgoztak. A gyártásban dolgozó tesztmérnökök kompetenciájához nem szükségszerűen tartozik hozzá a gyártott termék ismerete, sokkal inkább a tesztautomatáé, amivel dolgoznak. Így a forgatókönyvet megkapva beviszik a szükséges tesztlépéseket a mérőberendezés programozófelületére, majd az a tesztautomata szintjére fordítja azokat: definiálja a lépésenként szükséges, létrehozandó villamos összeköttetéseket, bemenetekre kapcsolandó értékeket, a teszt során felhasznált mérőkártyákat stb.

  A tesztek megalkotásához a gyártásban azonban nem elég a program elkészítése, hanem szükség esetén

---

[18] *Tesztforgatókönyv (test scenario): azon lépések meghatározása, amiknek egymást követve meg kell történnie a teszt során, tulajdonképpen a teszt program alapja.*

[19] *Teszteset (test case): bemeneti és kimeneti értékek kombinációja, amiket a tesztforgatókönyv során alkalmazunk.*

(új termékcsalád bevezetése) rendelni kell az új típushoz szükséges adaptereket is egy belső (gyárbeli műhely) vagy külső (méréstechnikai cég) résztvevőtől, hogy amikor a termelésben a próbagyártás céljából foglalt időablak elérkezik, adott legyen minden a tesztelők oldaláról is, hogy mindezt kipróbálják.

Látható, hogy a gyártás minden szintjén hangsúlyos szerepet kap a tesztelés a gazdasági szempontoktól a kész termék raklapra kerüléséig. Egy jól átgondolt folyamat nem létezhet megfelelő minőségellenőrzés nélkül, aminek segítségével lehetséges elviselhető szinten tartani a selejt-rátát, az abból fakadó költségeket elkerülve lehetővé tenni a vállalat nyereséges működését.

Fejlesztés –ötlettől a gyártási utasításig

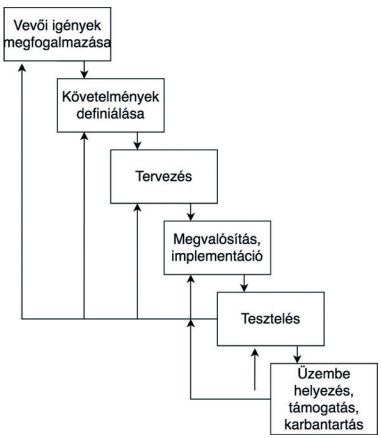
Amikor a megrendelőnek felmerül az igénye egy olyan termékre, ami még az adott formájában nem létezik, a gyártási folyamat beindulását a fejlesztés előzi meg. Ez számos formát ölthet, a termék jellegétől, a fejlesztő (továbbiakban beszállító) szervezeti felépítésétől függően: egész máshogy zajlik egy mobilalkalmazás fejlesztése egy egyszemélyes „garázsvállalkozásban", mint egy hardvert és szoftvert is magában foglaló, összetett készüléket szállító multinacionális vállalatnál.

Míg előbbi esetben nem szükséges a munkafolyamatok nagy felbontású definiálása, valamint a kísérő dokumentáció is lehet kevéssé részletes[20], addig utóbbinál gyakran az ipari szabványok is megkövetelik, hogy vertikálisan és horizontálisan is magas fokon dokumentált, jól átgondolt módon történjék a fejlesztési munka.

Természetesen a tesztelési tevékenységeket ismét nem érdemes önmagukban tárgyalni, azok céljait, módszereit megérteni úgy lehet, ha kontextusukban ismerjük meg azokat.

Fejlesztési életciklusok

Vízesés-modell



[20] *De sosem célszerű elhagyni, annak érdekében, hogy akár évekkel később is pontosan tudjuk, mit és miért csináltunk! (pl. garanciális követelések, vagy utólagos terméktámogatás esetén)*

A fejlesztési folyamatok legegyszerűbb változata a vízesés-modell, mely esetében a munka a vásárlói igények megfogalmazásával kezdődik. Ezekből az igényekből készülnek a fejlesztés alapját adó követelmények (más néven specifikációk), ahol a megrendelő kéréseit műszaki nyelvre fordítva meghatározzák a termék funkcióit, valamint azoknak az elvárt működését.

Fontos, hogy mielőbb, lehetőleg már a követelmények készítésekor bevonják mind a vevőt, mind a fejlesztési folyamat többi résztvevőjét, beleértve a tesztelőket is. Ily módon a lehető legkorábban lehetőség van az esetleges specifikáció-beli hibák, félreértések kiszűrésére, rengeteg olyan munkát megspórolva, ami ezen ellentmondások későbbi felszínre kerülése után akár komoly veszteségeket, késéseket okoznának a fejlesztési folyamatban.

A követelmények véglegesítését követően a tervezés következik, ahol a termék jellegétől függően meghatározzák a készülő szoftver architektúráját, valamint a hardver felépítését.

Ennek a kimenetelét (blokkvázlatok, lay-outok) felhasználva a megvalósítás, implementáció során készül el maga a termék, amit a tesztelés során ellenőriznek, hogy mindenben megfelel-e a követelményeknek. Az így talált hibákat, nem-megfeleléseket gyakran a vevővel közösen elemzik, hogy mennyiben befolyásolják a termék használhatóságát. Amennyiben egy ilyen nem-megfelelésről azt állapítják meg, hogy a termék használhatóságára gyakorolt hatása nem halad meg egy kritikus szintet, akkor annak a javításával csak akkor foglalkoznak, ha az általa igényelt erőforrások lehetővé teszi azt, de ha a hiba hatása átlépi ezt az „ingerküszöböt", az azt okozó munkafázist azonosítva javítják azt, majd ellenőrzik a megoldás helyességét[21].

Miután a teszteredmények alapján a termék késznek bizonyul az átadásra, ha a termék jellege megkívánja, az üzembe helyezés, a termék-támogatás, és a karbantartás kezdődik, ahol a tervezőasztalról kikerült szoftvert, vagy készüléket a valóság kereteihez igazítják, szükség szerint.

A vízesés-modell elsősorban egyszerűbb termékek és egy-, vagy néhányemberes projektek esetében alkalmazható eredményesen.

Előnye:

- Egyszerű, lineáris munkafolyamat.
- Pehelysúlyú (light-weight) dokumentáció.
- Könnyű menedzselhetőség, hiszen minden szakasza jól körülhatárolt tevékenységeket tartalmaz, és egyszerre egy lépés történik.
- Lehetővé teszi, hogy egy ember akár több fejlesztési szinten is tevékenykedjen egyszerre (pl. akár egy egyéni vállalkozó is végigviheti a megrendelés érkezésétől a termékkövetésig)

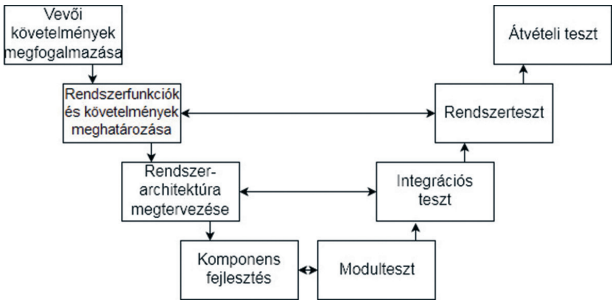
Hátránya:

- Csak a folyamat végén produkál kézzelfogható eredményt.
- Nehéz jól tervezni az egyes feladatok erőforrásigényét, váratlan akadályok tűnhetnek fel a projekt egyes pontjain.
- Körülményes és drága a sikertelen teszt után visszatérni a korábbi fázisokhoz, majd sok feladatot újra elvégezni.

---

[21] Példa: Egy konyhamérleg esetében észreveszik, hogy nem működik megfelelően a védelem fordított polaritású tápellátásra kapcsolás ellen, de a követelmények kimondják, hogy erre szükség van, és a megrendelő nem áll el ettől az igénytől. Ez esetben a tervezők visszakapják a munkát, hogy javítsák ki ezt a hiányosságot. Ezt követően újra megépítik az így módosított tervek alapján a mérleget, majd a tesztelők ellenőrzik, megoldódott-e a javítás, valamint validálják az összes funkciót, amit a módosítás érinthetett.

V-modell



A vízesés modell továbbgondolásaként is felfogható a V-modell, ami tulajdonképpen előbbi „félbehajtásából" jön létre. Alapját az a felismerés adja, hogy – mint az fentebb is megállapítást nyert – lehetőség van a korábbi fázisok megkezdésekor bevonni a fejlesztés későbbi fázisait, hogy korán felismerjék a lehetséges zsákutcákat, félreértéseket, a termék létrejöttét érintő kockázatokat.

A munka itt is a vevői igények megérkezésével kezdődik, majd a követelmények megalkotásával folytatódik. Ezt követően elkészülnek a termék részletes tervei, majd a megvalósítást követően a teszteléssel folytatódik.

Míg a vízesés-modell azonban egyszerűbb projektekre lehet optimális, függetlenül attól, hogy hardver- vagy szoftver alapú termékről beszélünk, a V-modellt elsősorban szofverfejlesztésben használják, és esetlegesen kiegészíthető a hardverfejlesztéshez, rendszerfejlesztéshez tartozó munkafázisokkal, majdnem minden esetben nagy összetettségű, biztonságkritikus termékek esetén.

Mint fentebb említésre került, a V-modell kulcseleme a későbbi munkafázisok bevonása az egyes fejlesztési tevékenységekbe. Minden fejlesztési fázisnak megvan a maga „párja" a validációs, tesztelői oldalon:

- A rendszerterfunkciók definiálásának a rendszerteszt
- Az rendszer architektúrális tervezésének az integrációs teszt
- A komponens fejlesztésnek a modul-, vagy unit teszt[22]

Az így bevont tesztelőknek lehetőségük van együtt dolgozni a fejlesztőkkel. Ennek a kedvező hozadékai a következők:

- A vevői követelmények elvi félreértésének csökkent kockázata specifikáció-felülvizsgálatok (továbbiakban review-k) által.
- A fejlesztői- és tesztelői környezetek nem ismeretéből fakadó, feleslegesen elégetett mérnök- és gépidők csökkentése (kevesebb félreértés).
- Magasabb intenzitású, hatásfokú információáramlás az egyes résztvevők között.
- Ún. test driven development megvalósulásának lehetősége az alacsonyabb szinteken (főleg komponens fejlesztés-modul teszt párosításban): egy új funkció implementációját az azt vizsgáló teszt kifejlesztése előzi meg. Amint ez megvan, a teszt lefut (és természetesen megbukik), majd ezt követően írják csak meg a funkciót megvalósító forráskódot. Bár első olvasatra furcsán hangozhat, ez a módszer különösen hatékony a követelményeket érintő félreértések minimális szintre csökkentésében,

[22] *Az egyes tesztelési szintekről olvashatunk A szoftvertesztelés szintjei című fejezetben.*

hiszen a teszt fejlesztése során a folyamat résztvevői mélyebben megismerik a követelményeket, és ennek a tudásnak a birtokában kezdődik csak maga az implementáció.

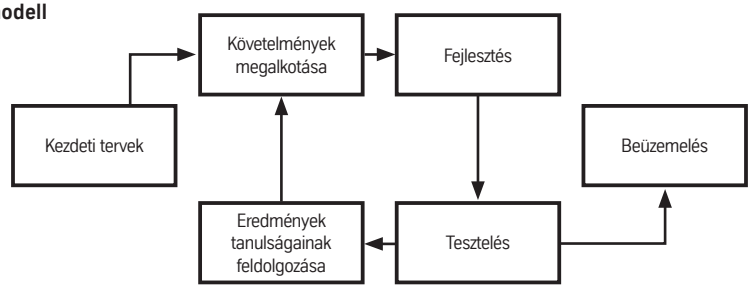
A V-modell előnyei:

- Átlátható, könnyen bevezethető (főleg, ha a vízesés-modellről vált erre egy fejlesztő cég).
- Korai hibafeltárást tesz lehetővé, ezzel elkerülve azok továbbgyűrűzését, és az ezekből fakadó, fokozott veszteségeket.
- Jól működik olyan környezetben, ahol a követelmények jól érthetőek, egzaktak és időben rendelkezésre állnak.
- A tesztelői aktivitás, a teszt fejlesztés megtörténhet a kód megalkotása előtt, amivel időt takaríthat meg a projekt (pl. a vízesés-modellel összevetve).

A V-modell hátrányai:

- Bizonyos helyzetekben kissé rugalmatlan (pl. az egyes munkafázisok elhúzódása időnyomást okoz a későbbi szakaszokban (ez a tesztelés szempontjából különösen nagy nehézségeket tud okozni)[23].
- Amennyiben egy változás történik az életciklus későbbi szakaszában, az egész projekt dokumentációját módosítani kell.[24]
- Korai prototípus hiánya.

Iteratív modell



A vízesés-modell mellett a hardverfejlesztésre leginkább jellemző metodológia az interatív fejlesztés. Ennek lényege a prototípus elkészítése minél gyorsabban a követelmények megalkotása után. Ennek hibátlansága ezen a ponton még nem cél, sokkal inkább az, hogy a valóságban lehessen megfigyelni az adott termék viselkedését. Ezt a követelmények ellenében történő, és kreatív (nem a specifikáció alapján, hanem tapasztalati úton megírt) tesztekkel lehet megtenni. A teszteket a termék újragondolása követi, az előzőleg szerzett tapasztalatok alapján.

---

[23]Ez a V-modell legnagyobb hátránya, ugyanakkor megfelelő menedzsmenttel elviselhető szintre csökkenthető a késések akkumulálódásából adódó időnyomás.

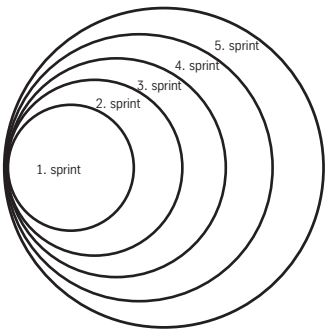
[24]Erre a problémára megoldást kínálnak olyan követelménykezelő szoftverek, amik ún. bilaterális követhetőséget adnak az egyes dokumentum-szintek között (pl. a vevői követelményekből link mutat a fejlesztők bemenetét adó termék-követelményekre és viszont, ahonnan az azt a követelményt fedő tesztre, ahonnan pedig annak a tesztnek az eredményére mutat egy-egy link). Mivel ezek a követelménykezelő szoftverek általában önmagukban is programozhatók, így megoldható, hogy egyetlen paraméter módosítása után az összes arra mutató linkkel rendelkező követelmény releváns értéke is módosuljon.

Ez jelenthet ténylegesen komponens-fókuszú hibajavítást, de adott esetben teljes újradolgozást is, függően a teszteredményektől. Azt a fázist követően, amikor a prototípus a validáció során megfelelt a követelményeknek, gyártásba kerülhet.

Az iteratív modell előnyei:
- Időben történő hibafeltárás lehetősége a prototípus korai rendelkezésre állása miatt.[25]
- Mivel az elején csak a magas szintű terveket tudjuk megvalósítani, a részletes technikai specifikációt pedig részben a termék valóságban produkált tulajdonságai határozzák meg, az alacsony szintű követelmények ritkábban elrugaszkodottak a valóságtól.
- A hangsúly az építésen van, és nem az adminisztráción.
- Gyors visszajelzés a megrendelőktől, a prototípusok bemutatásának köszönhetően.

Az iteratív modell hátrányai:
- Költséges, rendszerszintű hibák kockázata, hiszen a követelmények a projekt elején kevésbé tisztázottak, mint a korábbi modelleknél.
- Nincs lehetőség átlapolódásra az egyes iterációs fázisok között (a prototípus készültekor dől el csak néhány részlet, a tesztelők csak később vonódnak be a munkába).

Inkrementális fejlesztési modell



Az eddigi fejlesztési módszerek esetében a folyamat végén egy – ideális esetben – a vevői követelményeket száz százalékosan kielégítő végtermék kerül a megrendelőhöz. Gyakran előfordul azonban, hogy a készülő termékre sürgetőbb az igény, mint amennyi időt a teljes csomag elkészítése igényel. Főként a szoftverfejlesztés világában azonban lehetőség van arra, hogy – amennyiben a termék jellege ezt megengedi – a szerződésben foglalt funkciókat nem egyszerre, hanem időről időre, több lépcsőben kapja meg a vásárló, ahogy azokat sikerül megvalósítani.

---

[25]Ezt érdemes összevetni a V-modellel. Míg ott a tesztelők fejlesztésbe való korai bevonása teremtette meg a hibák gyors feltárásának lehetőségét, addig itt a prototípus gyors létrehozása: ugyanaz a cél elérése két, gyökeresen különböző módszerrel.

Fontos, hogy a szerződő felek a projekt indulásakor tisztázzák, milyen sorrendben, mely funkciók kerülnek kiszállításra a vevő felé. Amennyiben váratlan akadályok, nehézségek miatt az így megalkotott menetrendet mégsem sikerül tartani, azt a megrendelő és a beszállító szükség esetén újratárgyalhatja.

Azokat a fejlesztési életciklusokat, amik ilyen formában működnek, inkrementális modelleknek nevezzük. Közös jellemzőjük, hogy a szerződés tárgyát képező termék időegységről időegységre (ezeket általában sprintnek nevezik) új tartalommal bővül.

A sprintek önmagukban úgy foghatók fel, mint megannyi mini-projekt, egyedi azonosítóval, saját célokkal, menetrenddel, adminisztrációval. Egy sprinten belül a fejlesztők megvalósítják a tervezett funkciókat, feature-öket, amiket a tesztelők validálnak, majd a bizonyítottan a megrendelő igényeinek megfelelő funkciókat a már elkészült dolgokat tartalmazó „anya-termékbe" integrálják. Jellemző módon az egyes feature-öket a megrendelő preferenciái alapján kritikusság szerint priorizálják, az egymás után következő sprintekbe pedig ennek megfelelően kerülnek bele a funkciók. Az így gyarapodó rendszer működésének egyfajta végső ellenőrzéseként megvizsgálják azokat az aspektusait, amiket a bővítés a szakértők szerint érinthetett.[26, 27]

Az inkrementális fejlesztési modell előnyei:

- Rugalmas, hiszen az egyes feature-ök karakterisztikái szükség esetén az anya-projekt későbbi szakaszaiban is módosíthatók, amennyiben az még nem került kiszállításra.[28] Ez nagyobb szabadságot ad a megrendelőknek, amire sokszor szükség is van, mert a valóság nem mindig vág egybe a rendszer megálmodóinak elképzeléseivel, ily módon pedig lehetőség van a követelmények későbbi pontosítására a vevő oldali tesztek alapján.
- A termék bevezetése lépésről lépésre egyszerűbb a vevő oldalán, mint egyszerre megtanulni, vagy a felhasználókkal megtanítani egy komplex, teljesen új rendszert.
- Könnyebb – mind technikai, mind menedzsment oldalról – megvalósítani egy termék kifejlesztését, validációját és a vevőknek történő átadását több, kisebb lépésben, főleg komplex projekteknél.
- A vásárló számára kulcsfontossággal bíró feature-ök korai átadásával gyorsabban teremt üzleti értéket, hasznot a termék.
- Az anya-projekt élettartama alatt folyamatos a vevői visszacsatolás a termék működéséről, olcsóbb és könnyebb termékkövetés, -támogatás.

Az inkrementális modell hátrányai:

- A sprintek pontos megtervezésének alapfeltétele a kiterjedt gyakorlati tapasztalat az adott üzleti szegmensben, hogy az esetleges akadályokkal, hátráltató tényezőkkel (mindennel, ami a sikeres kiszállítás szempontjából kockázat lehet) már az induláskor tisztában legyenek.
- Az ipari tapasztalatok alapján a projekt teljes költsége drágább, mint a vízesés-, vagy a V-modell

---

[26]És egyéb más funkciókat is esetleg, amelyek a megrendelő számára kritikusak lehetnek. A tesztelendő feature-ök meghatározásáról bővebb leírás található A tesztelés életciklusa fejezetben.

[27]Ez az ún. regressziós teszt, amikor a szoftverben bekövetkezett módosítások alapján meghatározzák az érintett funkciókat, amiket a változások esetlegesen érinthettek, majd ezeket a szoftver publikálása előtt a már meglévő, és lefuttatott tesztek újra végrehajtásával ellenőrzik.

[28]Utóbbi esetben sem kizárt egy adott feature módosítása, azonban az már a megrendelő számára plusz költséget, kötbért jelent.

alkalmazása.[29]

- Fontos, hogy a fejlesztendő rendszer már a megrendelés pillanatában jól átgondolt legyen, hogy az egyes sprinteket eszerint lehessen tervezni.

Az inkrementális fejlesztési módszert alkalmazzák a modern ipar azon – megfelelő környezetben, jó megvalósítással hatékony – metodológiái, amiknek rövid bemutatása nélkül nem tárgyalható korszerűen a termékfejlesztés világa.

Az „agilis" módszertanok

Az ún. agilis módszertanok alapvetően az inkrementális modell továbbgondolásai, melyek lényegét hiba lenne elsősorban technikai oldalról megközelíteni. Gyakorlatilag ugyanazt csinálják, mint az előbbi, a szerződés tárgyát képező terméket időről időre új feature-ökkel látják el.

Ami mégis kiemeli őket, elsősorban szervezeti, működési kultúrájukban keresendő. Ennek mibenlétéről jó képet ad a 2001-es Kiáltvány az agilis szoftverfejlesztésért. [30]

„A szoftverfejlesztés hatékonyabb módját tárjuk fel saját tevékenységünk és a másoknak nyújtott segítség útján. E munka eredményeképpen megtanultuk értékelni:

- Az egyéneket és a személyes kommunikációt a módszertanokkal és eszközökkel szemben
- A működő szoftvert az átfogó dokumentációval szemben
- A megrendelővel történő együttműködést a szerződéses egyeztetéssel szemben
- A változás iránti készséget a tervek szolgai követésével szemben

Azaz, annak ellenére, hogy a jobb oldalon szereplő tételek is értékkel bírnak, mi többre tartjuk a bal oldalon feltüntetetteket."

Ezek a gondolatok az ipari szabványok szigora és a technológiák gyakori merevsége mellett alternatívát nyújtottak a fejlesztők számára.

Az agilis fejlesztés alapköve a fejlesztő csapat, amely szinte kivétel nélkül az összes ilyen szellemű módszertanban kevert kompetenciájú szakemberekből áll: vannak, akik fejlesztéshez, vannak, akik a teszteléshez értenek, és ők a sprintek során a lehető legnagyobb együttműködésben dolgoznak, támogatva a sikeres kiszállítást. Emiatt igen intenzív az információáramlás, és a fejlesztők és tesztelők közti félreértések kockázata minimális.

További közös kulcselem a megrendelői elégedettség fontosságának hangsúlyozása az egész folyamat során. Ennek érdekében a sprintek idejét minimalizálják (kettő, vagy akár egy hétre is), hogy a lehető legkevesebbet kelljen várni a vásárlónak az új feature-ökre, valamint a projekt teljes időtartalma alatt nyitottak a követelmények változására, akkor is, ha azok már régen megvalósított funkciókra vonatkoznak.

Az agilis módszertanok kívülről leglátványosabb tulajdonságainak többnyire egyáltalán nincs, vagy csak

---

[29]Azonban, ha belekalkuláljuk azt, hogy a megrendelő számára már a fejlesztés korai fázisában egy hasznot termelő terméket adunk, gyakran így is megéri neki a többletköltség.

[30]http://agilemanifesto.org/iso/hu/manifesto.html

minimális a műszaki tartalma, kulturális viszont annál is több: ilyen a személyes kommunikáció előnyben részesítése az írottal szemben, a dolgozók és a vezetők közötti informális viszony, a rendszeres számonkérés, ami egyúttal lehetőség a segítségnyújtásra is, valamint az, hogy az agilis módszerek metrikája elsősorban a termék működőképességére koncentrál, semmint az adminisztrációra.

Bár az itt leírtak alapján könnyen az gondolható, hogy az agilis módszertanok a fejlesztés lehető legfejlettebb metodológiái, fontos megjegyezni, hogy ezeket félreértve, nem megfelelően alkalmazva könnyen okozhatnak jóvátehetetlen károkat egy projekt sikeressége szempontjából. A dokumentáció itt is fontos, csak úgy, mint a folyamatok betartása (igaz, nem minden áron), ezek elhanyagolása ellehetetleníti a termék esetleges karbantartását, követését. A teljes mértékben a vevőre való hagyatkozás pedig – ha nem vigyázunk – elhomályosíthatja a projekt eredeti célját a gyakori irányváltások miatt, komoly károkat okozva ezzel a fejlesztőknek és a megrendelőnek is, így feltétlenül javasolt a vásárlói igények megvalósíthatósági költségeinek figyelembe vétele, a vevő bevonásával.

Az itt bemutatott fejlesztési modellek mindegyike garantálhatja a projekt sikerét, amennyiben azt megfelelően átgondolva alkalmazzák hozzáértő emberek, olyan környezetben, amire az való. Ezen felül általában semmi nem tiltja az egyes módszerek elemeinek ötvözését annak megfelelően, amire a vállalatnak ott és akkor szüksége van. A fejlesztés technológiája éppoly bonyolult, mint a gyártásé, megalkotása tapasztalt szakembert kíván, aki tudja, mit mivel célszerű elérni.

Szoftvertesztelés

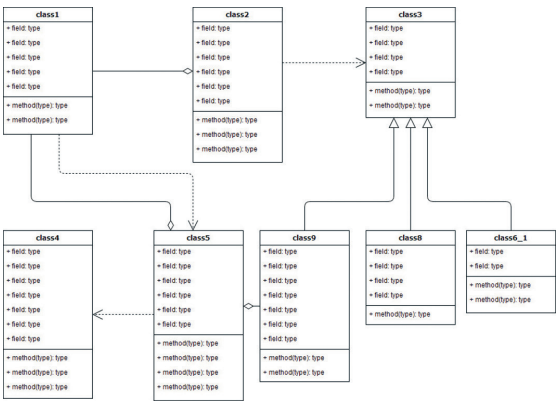
A hardverfejlesztés során alkalmazott tesztelés tulajdonképpen nem különbözik a gyártás során alkalmazott verifikációktól, legfeljebb részletességében, a tesztpontok számosságában, ezért a továbbiakban e jegyzet a szoftvertesztelésre koncentrál.

A szoftverek ellenőrzésének alapját ugyanazok a követelmények adják, amik a fejlesztők bemenetéül szolgáltak. A tesztelő ezeken a követelményeken halad végig, ellenőrizve, hogy a szoftver az egyes pontoknak megfelel-e, rögzítve az eredményt és szükség esetén adminisztrálva a talált hibát, nem-megfelelést.

A szoftvertesztelési feladatokat csoportosíthatjuk a megközelítésük alapján, valamint az alapján, hogy milyen szinten helyezkednek el a szoftver életciklusában.

A szoftvertesztelés szintjei

Unit-, modul-, vagy komponens teszt



A szoftver architektúrája modulokra, elemekre bontja a teljes programot. Ezek az elemek (unitok) egy-egy részfunkciót tartalmaznak, és a szoftver legkisebb egységeinek tekinthetők. A közöttük zajló interakciók a tervezők (architect-ek) által kerültek meghatározásra. Minden modul adatokat fogad egymástól, és azokat feldolgozva továbbküldi a többi komponensnek. Az egyes elemek ilyen együttműködésének eredménye a tulajdonképpeni kész szoftver.

A modulteszt során a tesztelő ezekre az elemekre koncentrál. Az architektúrában definiálva van, milyen beérkező adatokkal milyen műveleteket kell elvégezni, és azokat milyen formában, melyik modul felé kell eljuttatni.[31]

---

[31] *Gyakorlati példa a szoftvermodulra: egy szenzorkezelő kliens egy háromtengelyű gyorsulásérzékelő adatait továbbítja a szóban forgó modulunk felé, aminek az a funkciója, hogy a beérkező szenzoradatokból kiszámolja a mozgásvektor különböző irányú összetevőit. Emellett megkapja ugyanettől a komponenstől a szenzor fizikai státuszát (MEGBÍZHATÓ, NEM ELÉRHETŐ, vagy valamilyen hardveres hiba miatt NEM MEGBÍZHATÓ). Amennyiben a szenzor jele MEGBÍZHATÓ, továbbítja a kiszámolt vektor összetevőket három, ezt felhasználó komponensnek (egy döntéskezelő algoritmusnak, egy belső diagnosztikai, és egy, a külső kommunikációért felelős egységnek, ami egy vizuális megjelenítő felé kommunikálja az éppen aktuális mozgásadatokat). Amennyiben a szenzor jele NEM ELÉRHETŐ, vagy NEM MEGBÍZHATÓ, a szóban forgó modul jelzi ezt az általa címzett komponensek felé.*

A szoftvertesztelés ezen szintje a modulok közti kommunikációval nem foglalkozik, célja magának az adott komponensnek a működésének vizsgálata. Ezt úgy tudja elérni, hogy önmagában, szeparált szigetként vizsgálja az egyes modulokat. Ahhoz, hogy ezeket így, a környezetükből kiragadva is működésre bírhassa, egy virtuális tesztkörnyezetet (framework-öt) alkot, amiben a tesztelt szoftverkomponens úgy „érezheti", koherens rendszerben van.

A virtuális tesztkörnyezet eszközei[32]:

- Stub: A szoftverfejlesztés során felhasznált olyan kódrészlet, ami a végleges és működő rendszer egy adott funkciójának kiváltására szolgál. Pl. ha a kész programban létezik egy giveMeTheName(dataBaseIndex) függvény, ami egy külső adatbázis megfelelő helyén tárolt nevet kérdez le a vizsgált komponensben, stub-ként annyit fog tartalmazni, hogy visszaad egy előre (akár a tesztmérnök által, valós időben) definiált szöveget. Ez azért fontos, mert sokszor szükséges olyan funkciók vizsgálata, amik nem az adott modulban található elemektől kapják a bemenetüket, de ezeket a lekérdezéseket, függvényeket ki lehet cserélni az adott stub-bal, megőrizve a komponens szeparáltságát, valamint így helyettesíthetők időlegesen olyan funkciók, amiknek a tényleges implementációja még nem készült el.
- Fake, mock-object: Mivel egy komponens a bemeneteire érkező adatokat alakítja át az architektúrában definiált módon, a működésének teszteléséhez elengedhetetlen, hogy a bemeneteire érkezzen adat. A tesztframework fejlesztői ezt úgy oldják meg, hogy fake-eket, mock-objecteket hoznak létre, amik olyan kódrészletek, amik a modul bemeneteire eljuttatják a szükséges (általában a tesztmérnök által, valós időben) definiált adatokat, környezeti változókat, ezzel szimulálva a valós rendszer viselkedését.
- Test driver: Arra hivatott, hogy szimulálja a magasabb szintű komponensek viselkedését, azáltal, hogy azoknak a kimeneteit helyettesíti oly módon, hogy a rendszer helyes viselkedése ezáltal igazolhatóvá váljon. Pl. a magasabb komponens elérését az alacsony szintű komponens által, valamint az annak történő helyes érték átadását egy változó nevezetes értékének felvételével indikálja.
- Test harness: Az automatizált tesztframework és a megalkotott tesztesetek együttesét jelenti (ide tartoznak az előbbi eszközök is).

A modul teszt célja tehát az adott komponens előírások szerinti viselkedésének igazolása. Nem ritkán maguk a szoftverfejlesztők végzik, nem szükségszerűen igényel független tesztelőt. Nagyon fontos, hogy a tesztek megőrizzék a szoftver-egység szeparációját, mert ha ez nem sikerül, a hibafeltárás nem lehet sikeres, hiszen a vizsgálat látótere más részegységekre is kiterjed, nagyon megnehezítve a talált problémák hatékony elemzését.

A test driven development leginkább ezen a szinten alkalmazható: a követelményeket olvasva a fejlesztők előbb a komponens tesztjét tervezik meg. Ezt követően kifejlesztik a hozzá szükséges test harness bővítményeket (framework frissítés, ha szükséges, valamint tesztesetek megalkotása), majd csak ezután kezdik el implementálni a releváns funkciókat. A modulteszt a kódolás kezdetekor elindul, a komponens akkor tekinthető elkészültnek, amikor a tesztje sikeresen átmegy („zöldre fut").

---

[32]Ezek az eszközök a szoftvertesztelés összes szintjére igazak lehetnek.

Integrációs teszt

A modultesztek elkészülte után a következő lépés a szoftver integrációjának ellenőrzése. Ezen tesztek célja azoknak a hibáknak a megtalálása, amik az egyes modulok hibás együttműködéséből adódnak (a nem megfelelően kezelt/megvalósított interfészek[33] miatt).

Integrációs teszt validálja továbbá azokat az összetett rendszereket is, amik további alrendszerekből, pl. több szoftverből állnak. Ebben az esetben az egyes szoftverek közötti interakciók hibáinak feltárása a cél.

Ezeknél a teszteknél a szoftver egészét vizsgálják, a követelményekben leírt interfész-definíciók alapján. A bemeneti értékek, valamint az interfészek működése szempontjából érdekes szoftveres változók különböző kombinációival vizsgálják, hogyan változnak a különböző kimenetek, valamint egyes belső változók értékei. Ezeket a teszteseteket úgy határozzák meg, hogy az interfész-specifikációk egészét lefedjék.

Az integrációs tesztek lehetséges megközelítései a következők:

- Big Bang: a rendszer egészének fejlesztőktől történő átvétele után kezdődik el az integrációs tesztelés. Jogosan merülhet fel a kérdés, hogy voltaképpen mi a különbség eközött, valamint az ezt a szintet követő rendszerteszt között? Első pillantásra nem is annyira jelentős, hiszen egyes rendszerfunkciókat elképzelhető, hogy integrációs teszttel ellenőriznek, amennyiben az implicit módon az egyes interfészek közötti helyes működést igazolja. A rendszerteszt azonban a felhasználó számára leginkább látható, funkcionális és nem funkcionális tesztekkel foglalkozik, míg az integrációs teszt a komponensek együttműködését vizsgálja, valamint figyelmet szentel a rendszer – akár kódszintű – felépítésének, architektúrájának is.
- Top Down (felülről lefelé): ez a megközelítés a magasabb szintű komponensek tesztelésével kezdődik, és az alacsonyabb szintűek ezután következnek. Ez lehetővé teszi, hogy megkezdhessék az integráció vizsgálatát, amíg az alacsonyabb szintű modulok elkészülnek, addig pedig stub-okkal helyettesítik azokat. Alkalmazásához elengedhetetlen feltétel azonban, hogy a rendszer architektúrája végleges és jól definiált legyen, máskülönben a már integrált komponensek változása miatt meg kell ismételni a teszteket, és a munka hiábavaló volt.
- Bottom Up (alulról felfelé): ebben az esetben – ellentétben az előzővel – az alacsonyabb szintű komponensek integrációját vizsgálják meg először, majd ezt követi a magasabb szintűek bevonása a tesztelésbe. A még meg nem lévő magasabb szintű egységeket test driverekkel helyettesítik. Az előbbihez hasonlóan itt is szükséges a szoftver architektúrájának végleges mivolta.
- Hibrid/Szendvics: ez a megközelítés a fentiek kombinációja az egyes komponensek kritikusságának függvényében (amennyiben a projekt sikerében alacsony és magas szintű komponensek is kulcsfontosságúak, ezek elkészülte és integrációja elsőbbséget kell, élvezzen a kevésbé kritikus, vagy kevesebb kockázatot hordozókéval szemben).

Az integrációs teszteket a modultesztekkel ellentétben független tesztmérnök végzi, vállalaton belül, de egyes esetekben akár külső cégtől is.

---

[33]Interfész: egy számítógépes rendszer (esetünkben: maga a szoftver) egyes komponenseinek határa, ahol információcsere zajlik. Hookway, B. (2014). „Chapter 1: The Subject of the Interface". Interface. MIT Press. pp. 1–58. ISBN 9780262525503.

Rendszerteszt

Ahogy az integrációs teszt esetében, a rendszerteszt alatt is az integrált szoftver áll a figyelem középpontjában.

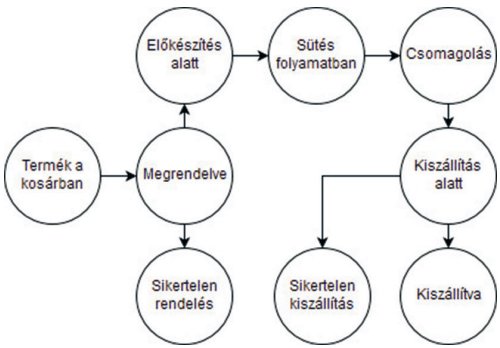
Előbbivel ellentétben azonban itt nem foglalkoznak a programkód és a rendszerarchitektúra felépítésével, logikájával, hanem csak és kizárólag annak viselkedésén, felhasználói élményén, funkcionalitásán van a hangsúly.A szoftvert minden esetben olyan hardveren tesztelik, amin az végül éles használat közben is futni fog.

Az itt használt tesztesetek bemeneti paraméterei azok a környezeti változók, amik a tényleges szoftver bemenetei is, legyen az a grafikus felület egy checkboxa, szöveg mezője, egy hőmérő, vagy más szenzor jele. Az előzőekkel eltérően itt a leggyakoribb a manuális tesztelés, abban az esetben ugyanis, amikor egy fejlesztő cég több, különféle jellegű terméket fejleszt egyedi megrendelések teljesítésére, nem mindig gazdaságos az automatizált tesztframework kifejlesztése.

A rendszertesztek vizsgálhatják a szoftver funkcionális, és nem-funkcionális viselkedését. Előbbi esetben az ellenőrzés a követelmények ellenében történik, sorról-sorra, objektumról-objektumra véve azokat a pontokat, amik a projektdöntés és a technikai lehetőségek miatt ezen a szinten tesztelendők. A nem-funkcionális, vagy kreatív tesztek során olyan dolgokat vizsgálnak, amik bár nincsenek írásba adva a specifikációkban, a felhasználói élmény, és a stabil működés szempontjából mégis ronthatják a termék értékét.

Példák rendszertesztekre

1. Funkcionális tesztek:
   a. Egy népszerű pizzarendelő portál esetében a követelményekben szerepelt, hogy nyomonkövethető kell legyen az online megrendelés állapota. Az ebben lehetséges átmenetek is meghatározásra kerültek. A teszt megkísérli átállítani egy virtuális megrendelés állapotát a megengedett, valamint a meg nem engedett átmenetekkel.



   b. Egy mosógép vezérlőszoftverének észlelnie kell, ha túl sok ruhát pakoltak bele. A mosógépgyár fejlesztőközpontjában az erre a projektre kirendelt rendszerteszt-mérnök belepakol a mosógépbe a kritikus súly alatti ruhát, majd rögzíti jegyzőkönyvbe a gép reakcióját (jelez-e, hogy sok a súly?), majd még egy súly elhelyezésével túlterheli azt, ismét rögzítve a

jegyzőkönyvbe a gép reakcióját.

c. Egy repülőgép szellőztetőjének vezérlőprogramjában belső diagnosztikai funkciók vannak, amik a rendszer szabványban vállalt biztonságosságát hivatottak szolgálni. Az egyik ilyen diagnosztikai funkció feladata észlelni a füstérzékelők vezérlőjének jeleiből, hogy a repülőgépen tűz ütött ki. A diagnosztikának a követelmények szerint jeleznie kell, ha a füstérzékelők vezérlője a gép bármely pontjáról tüzet jelent, és egy meghatározott vészhelyzeti protokollt kell indítania, ami elzárja az oxigént a tűz forrásától, és széndioxiddal oltani kezdi azt. A tesztmérnök a tesztframework környezeti paraméterei közül kiválasztja a tűzvezérlő jeleit, majd az összes lehetséges kombinációban kipróbálja azt, rögzítve a vizsgált szoftver viselkedését.

2. Nem-funkcionális tesztek:
   a. Egy webshop szerverére bejelentkezik egyszerre 3 millió virtuális felhasználó. A tesztmérnök megvizsgálja, mennyire lassult be a szerver válaszideje, és mennyiben befolyásolja ez a webshop használhatóságát.
   b. Az ismert online számítógépes szerepjáték legújabb kiegészítőjében megújul a felhasználói interfész. Tesztelők ellenőrzik a megváltozott UI használhatóságát.
   c. Egy gépkocsi ablaktörlővezérlő szoftverét 3 óra alatt 400-szor ki-be kapcsolják, majd órákon át törlik vele az autó ablakát, szárazon. Ez alapján meggyőződnek róla, mennyire bírja a termék a hosszútávú, intenzív terhelést, használatot.

A rendszerteszteket erre specializált, független[34] tesztelők végzik a vállalaton belül, vagy szükség/vevői igény szerint külső megbízott cégtől.

Átvételi teszt

Az átvételi tesztek technikai szempontból nem sokban különböznek a rendszertesztektől. Fókuszukban ugyanúgy a felhasználói élmény és a használhatóság áll. Ezeket a teszteket minden esetben a fejlesztő vállalaton kívülről érkezett tesztelő, vagy potenciális végfelhasználók egy – erre a célra szerződtetett – csoportja végzi.

A megrendelő az ezzel megbízott céggel együtt azonosítja azokat a kulcsfontosságú funkciókat, amiknek a működését ellenőrizni kell, a tesztek ezekre fókuszálnak. Ez a típusú átvételi tesztelés elsősorban az ipari-, és üzleti szoftverek esetében bevett gyakorlat.

Amikor a lehetséges felhasználók végzik a tulajdonképpeni tesztelést, általában szórakoztatóipari szoftverről beszélünk. Ezt, amennyiben a fejlesztő cég helyszínén zajlik, alfa tesztelésnek, amennyiben a felhasználók otthonában történik, béta tesztelésnek hívjuk.

---

[34] *A függetlenség egyébként a tesztelés egésze során is fontos, hiszen ez elengedhetetlen a vizsgált rendszer működésének objektív megítéléséhez.*

A szoftverteszt fejlesztés megközelítései

A szoftvertesztelés alapvetése, hogy az ún. kimerítő tesztelés[35] a gyakorlatban megvalósíthatatlan, mert az gazdaságilag és a projekt határidő szempontjából tarthatatlan lenne. Emiatt a tesztelő arra kényszerül, hogy különböző technikákkal oly módon csökkentse le minél alacsonyabbra a tesztesetek számát, hogy a tesztek a projekt számára épp elégséges visszajelzést nyújtsanak a szoftver minőségéről, azonosítsák azokat a hibákat, amik a megrendelés teljesítése szempontjából kritikusak lehetnek. E fejezet ezen tesztfejlesztési technikákat, megközelítéseket ismerteti.

Statikus tesztelés

Statikusnak nevezzük azokat a tesztelési technikákat, amik során az ún. szoftveres work product-okat[36] manuálisan, vagy valamilyen céleszközzel, toollal vizsgálják, de maga az ellenőrzött programkód nem kerül futtatásra. A szoftver életciklusának korai stádiumában kezdődik, és egészen a projekt végéig tart.

Statikus technikák:

- Review: a review során a projekt résztvevői a vállalat különböző területeiről (fejlesztés, tesztelés, adminisztráció) együtt kísérlik meg azonosítani és megtalálni a potenciális hibákat a szoftver követelményeiben, amik később bármilyen formában (félreértés, tovább gyűrűző hiba) akadályozhatják a szoftver sikeres kiszállítását. Léteznek formális és kevésbé formális review technikák, előbbiek meghatározott keretek között zajlanak, míg utóbbiak egyszerűbb megbeszélések. Fontos, hogy a formális review-król minden esetben dokumentáció készüljön, és az esetleges találatokra megoldást keressenek.
- Walk through[37]: Kevésbé formális technika, amit a követelmények alkotói vezetnek. A résztvevőkkel végignézik a szóban forgó dokumentumot, és együtt megértik annak tartalmát. Ez a technika különösen azoknak lehet hatékony, akik nem szoftveres háttérrel rendelkeznek, elkerülendő az esetleges félreértéseket.
- Inspection[38]: Formális review technika, képzett moderátorok vezetik. Az inspectionről saját dokumentáció készül, amit a benne résztvevő reviewerek részletesen megismernek és saját hozzáadott ötletekkel bővítenek, még a találkozó előtt. Az itt talált hibákat adminisztrálják, és akciótervet készítenek a kijavításukra. Ezt utóbb egy ún. follow-up értekezlet alatt ellenőrzik. Az inspection célja a vizsgált követelmények készítőjének segítsége konstruktív javaslatok által, valamint, hogy a követelményekben talált hibákat olyan korán megtalálják és javítsák, amilyen korán az csak lehetséges. Ezáltal megelőzhető a később továbbgyűrűző hibák magasabb költsége, a közben zajló hatékony, élő információcsere pedig a projekt résztvevőinek mélyebb megértéséhez járul hozzá, növelve a későbbi implementáció és a tesztek hatékonyságát.

---

[35] *kimerítő tesztelés (exhausting testing): amikor a létező összes bemeneti paraméter létező összes kombinációját kipróbáljuk*
[36] *Software Work Product: a szoftverfejlesztés során keletkező dokumentációk és forráskódok össszessége.*
[37] http://istqbexamcertification.com/what-is-walkthrough-in-software-testing/
[38] http://istqbexamcertification.com/what-is-inspection-in-software-testing/

Dinamikus tesztelés

A dinamikus tesztelés során a szoftver működés közbeni viselkedése kerül ellenőrzésre. A forráskód elkészültét követően kezdődik, és az átvételi tesztekig folytatódik. Ennek során a bemeneti értékek egy adott kombinációja mellett a szoftver viselkedése kerül kiértékelésre, általánosságban elmondható, hogy ez az a technika, amire egy laikus a szoftvertesztelés alapján elsőre gondol.

White Box megközelítés

Amennyiben a szoftver vizsgálata során annak szerkezete, strukturalitása áll a figyelem középpontjában, a megközelítést white box (fehér doboz) tesztelésnek hívjuk, a szakirodalomban és az iparban azonban találkozhatunk ugyanezzel glass box, clear box, transparent box, vagy structural testing néven is.

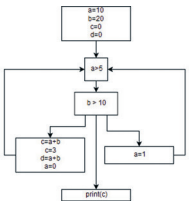
Ide tartozik az összes olyan tesztelési technika, ami során a követelmények, a forráskód, vagy a szoftver architektúrája kerül ellenőrzésre. Ez a megközelítés elsősorban a modul- és az integrációs tesztek során kerül alkalmazásra.

A white box tesztelés során alkalmazott módszerek esetében szükséges, hogy a tesztmérnök mélyrehatóan ismerje a vizsgált forráskódot. A tesztesetek megalkotásakor figyelembe kell venni a szoftver működését, hogy azok az algoritmusok minden lehetséges útvonalát érintsék, visszajelzést adva azok működésének helyességéről.

White Box tesztfejlesztési technikák

- Control flow tesztelés[39]: statikus tesztelés, melynek során a kód komponensei grafikusan megjelenítődnek egy control flow grafikon formájában. A tesztesetek ezt követően úgy kerülnek megalkotásra, hogy az ebbe a grafikonban szereplő egységeket lehetőség szerint 100%-ban lefedjék. Ennek mérőszáma a coverage, ami az így, tesztesetekkel lefedett objektumok arányát mutatja az összes objektumhoz képest. Ezek alapján megkülönböztetjük a következő control flow teszteket:
    - Statement coverage: a control flow tesztelés legalsó szintje. A tesztek során ténylegesen végrehajtott statement[40] arányát mutatja a forráskódban szereplő összeshez viszonyítva.
    - Decision/branch coverage: a program algoritmusában szereplő összes lehetséges döntés, és a tesztek során választott döntések arányát mutatja.

  Fontos, hogy bár a control flow tesztelés alapvetően statikus, hiszen a lehetséges program-végrehajtási útvonalak meghatározásához nem kell futtatni a szoftvert, ugyanakkor az itt keletkezett adatok a tesztesetek megalkotása, majd azok végrehajtása, futtatása során hasznosíthatók, hogy ily módon növeljék a szoftver minőségéről alkotott kép felbontását.



[39] http://blog.qatestlab.com/2016/03/07/control-flow-testing/
[40] Tulajdonképpen programsor.

- Data flow tesztelés: a data flow tesztelés során a tesztesetek megalkotásánál azokra a pontokra fektetik a hangsúlyt, ahol egy adat az adott komponensbe (vagy az egész rendszerbe) megérkezik, és azt vizsgálják, hogy ennek az adatnak a későbbi felhasználása, továbbadása megfelel-e a szoftver követelményeinek, az interfészek és/vagy a modulok funkcionális definícióinak. A data flow tesztelés a következő hibákat segít megtalálni:
    - o egy változót deklarál, de később nem használ a program
    - o olyan változót próbál meg használni, ami azelőtt nem került dekralárásra
    - o egy változót többször is deklaráltak, mielőtt felhasználásra került
    - o a változó törlésre kerül, mielőtt bárhol is felhasználnák

  A data flow tesztelés elsődleges felhasználási területei a modultesztek és az integrációs tesztek, mert míg előbbi esetében a bemenetek felhasználásának helyessége, utóbbi esetben a komponensek közötti interfészek működésének helyessége mérhető fel.
- Kód review: ide tartozik tulajdonképpen a kód review is, amikor a fejlesztők felülvizsgálják az egymás által implementált forráskódot, lehetőséget biztosítva a hiba korai megtalálására.

Black Box megközelítés

Amikor egy rendszert úgy vizsgálunk, hogy nem vesszük figyelembe annak szerkezetét, technikai megvalósításait, hanem „fekete doboznak" tekintjük, lehetővé válik a fókuszt teljesen a funkcióira irányítani. Ezt black box megközelítésnek hívjuk, melynek során az ellenőrzött szoftver kimeneteit kizárólag a bemenetek függvényeként fogjuk fel[41].

A black box tesztelés alkalmazható a korábban említett összes minőségellenőrzési szinten, modul-, integrációs-, rendszer- és átvételi teszten is. A white box megközelítéssel szemben az ezzel foglalkozó tesztmérnöknek nem szükséges a szoftverkód ismerete, sőt, amennyiben a tesztkörnyezet megengedi, elképzelhető, hogy még programozási tudás sem[42]. A black box tesztelés egyedül a követelményekre koncentrál, arra, hogy a vizsgált rendszernek mit kellene csinálnia, s nem arra, hogyan (a szoftver architektúrát, a benne foglalt komponens- és interfész definíciókat a tesztesetek elkészítése során nem veszik figyelembe).

Black Box tesztfejlesztési technikák

- Decision table tesztelés: amikor több bemenet kombinációi határozzák meg a rendszer kimenetét, célszerű egy, az összes érintett bemenetet és a kérdéses kimenetet, vagy kimeneteket tartalmazó mátrixot, táblázatot felvenni, ami tartalmazza előbbiek lehetséges kombinációit, és a követelmények alapján az adott bemeneti kombinációk mellett a kimeneten megjelenő értékeket. Ezt követően a tesztesetek megalkotása viszonylag egyszerű, a táblázat egy oszlopa egy tesztesetet jelent.

---

[41] Bár a szakirodalom e megközelítést pusztán a szoftvertesztelés témakörében tárgyalja, érdemes elgondolkodni azon, hogy a gyártásbeli végtesztelés is ide tartozik, hiszen akkor már a rendszer konkrét felépítése nem kerül célkeresztbe, csak a munkadarab, termék funkcionalitása.

[42] Pl. manuális tesztnél, vagy olyan, magas szintű, automatizált teszt framework esetében, ahol egy grafikus felület segítségével el lehet készíteni a tesztforgatókönyveket, és egy táblázatban meg lehet adni a teszteseteket. Itt nem szükséges egy programnyelv ismerete (mert azok, akik ezt a framework-öt készítették, elég, hogy értenek hozzá).

Bemenet	Teszteset1	Teszteset2	Teszteset3
Felvenni kívánt összeg <= Bankkártya balansz	IGAZ	HAMIS	HAMIS
Hitelkeret biztosított	-	IGAZ	HAMIS
Kimenet			
Sikeres pénzfelvétel	IGAZ	IGAZ	IGAZ

- All-pairs/Pairwise tesztelés: előbbihez hasonlóan az all-pairs tesztfejlesztési technika is egy táblázat megalkotásával határozza meg a lefuttatandó teszteseteket, azonban ez egyúttal azok számának esetenként radikális csökkentésével is jár. Célja, hogy a tesztesetek úgy legyenek megalkotva, hogy a bemenetek közül bármelyik kettő közül vizsgálatra kerüljön az összes lehetséges kombináció, anélkül, hogy az összes lehetséges kombinációt le kelljen futtatni.

Életkor	Neme	Családi állapot
Valid [0;120]	Férfi	Házas
Valid [0;120]	Nő	Egyedülálló
Invalid <0	Férfi	Egyedülálló
Invalid <0	Nő	Házas
Invalid 120<	Férfi	Egyedülálló
Invalid 120<	Nő	Házas

A fenti példa egy weboldal olvasói statisztikákat készítő kérdőívéhez tartozik. Három adatot kértek a felhasználóktól, életkorukat, nemüket és családi állapotukat. Az életkor egy szövegdobozt tartalmaz, ami numerikus bevitelt tesz lehetővé. A nem és a családi állapot egy ún. rádiógomb (radio button) által dönthető el, aminek két állapota lehet, Férfi/Nő, és Egyedülálló/Házas. A tesztelők az életkornak három lehetséges intervallumát határozták meg, amik esetében a kimenet a teljes szakaszon megegyezik (lásd később: ekvivalencia osztályok): 0 és 120 éves korig elfogadott (valid) a tartomány, míg két nem elfogadott (invalid) intervallum létezik, a negatív számok, és a 120 évnél nagyobb számok halmaza (a követelmények meghatározták, hogy a kérdőív ezeket a válaszokat ne fogadja el). A három bemeneti változó lehetséges összes kombinációja 3*2*2=12.

A pairwise tesztelés során kiválasztották azt a két változót, aminek a legtöbb lehetséges változata van. Ezek az életkor(3 lehetséges intervallum), valamint, mivel a másik kettőnek, a nemnek és a családi állapotnak 2-2 lehetősége van, ezért tetszőlegesen az előbbi. Ennek megfelelően e két bemenet összes kombinációját (3*2) vették. A harmadik oszlopot ennek megfelelően kezdték feltölteni. Az első sorba a házas, míg a másodikba az egyedülálló állapot került. A harmadik-negyedik sorban ezt a két értéket megcserélték, mert így érhető el, hogy a Nem és a Családi állapot összes kombinációja is kipróbálásra kerüljön. Ezt követően tetszőleges kombinációval feltölthető az ötödik és a hatodik sor, a lényeg, hogy e kettőben mindkét lehetőség szerepeljen, hogy ily módon a Kor harmadik lehetséges értéke is kipróbálásra kerüljön a Családi állapot minden lehetséges elemével.

Ily módon a vizsgált tesztesetek száma rögtön a felére, hatra csökkent (nem beszélve arról, ha az összes elfogadott életkort kipróbáltuk volna). Ezáltal jelentősen csökkent a tesztek futásidejének a nagysága.

- Ekvivalencia osztályok használata (equivalence partitioning): a szoftver bemenetei gyakran feloszthatók

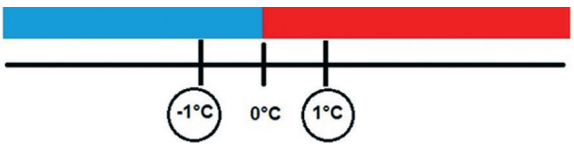
olyan tartományokra, amelyek mindegyike esetén ugyanaz a kimeneti érték/állapot várható. Amikor a teszteseteket ezek figyelembevételével alkotják meg, az az alapelv, hogy a teszteknek érinteniük kell ezen ún. ekvivalencia osztályok legalább egy elemét.

Az alábbi példa egy népszerű társkereső regisztrációs felületén lévő ellenőrzés lehetséges bemeneteinek ekvivalencia osztályait mutatja. A követelmény szerint csak 18. életévét betöltő személy veheti igénybe a szolgáltatást, a legnagyobb megengedett életkor 120 év.

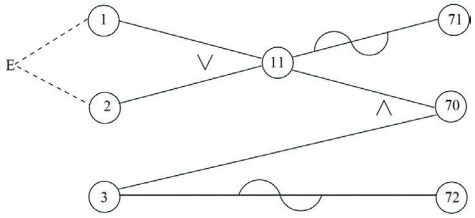
Életkor	Regisztrálhat
Valid [18;120]	IGEN
Valid [18;120]	IGEN
Valid [0;18]	NEM
Valid [0;18]	NEM
Invalid <0	NEM
Invalid <0	NEM
Invalid 120<	NEM
Invalid 120<	NEM

Látható, hogy itt az első ekvivalencia osztály 18-120 évig tart (az emberek várható életkorát figyelembe véve került meghatározásra ez az érték). Aki ezen a koron belül van, az regisztrálhat az oldalra, a rendszer tovább lépteti. Van egy másik értelmezett tartomány, 0-18 évig. Belátható, hogy kiskorú személyek is megpróbálhatják használni az oldalt, nekik azonban a szabályzat szerint még várniuk kell a nagykorúvá válásukig. Végül az előző példához hasonlóan itt is van két nem elfogadott tartomány, a negatív életkorúak, valamint a 120 éven felüliek.

- Határérték analízis (boundary value analysis): amikor egy kimenet úgy került meghatározásra, hogy a bemenet egy meghatározott értékének két oldalán lévő tartományokban máshogy kell viselkednie (pl. a víz 0 Celsius fok alatt megfagy, míg felette folyékony halmazállapotú), úgy eszerint a megközelítés szerint elegendő a nevezetes határérték alatti és feletti értékekkel megvizsgálni a szoftver működését.



- Ok-okozat diagram (cause-effect graph): ebben az esetben a bemeneteket és a kimeneteket grafikusan jelenítik meg a tesztelők, előbbiek okként, utóbbiak okozatként szerepelnek, esetleges közbülső objektumokkal (pl. a bemeneteket összekapcsoló logikai összefüggések). Ez a megoldás segíthet csökkenteni a decision table sorait, vagy jobban megérteni a rendszer működésének összefüggéseit.



- Error guessing: amikor egy fejlesztőcégnek egy adott jellegű termék előállításában tapasztalata van, a korábbi projektek tanulságai alapján a tesztelőknek lehetnek elképzelései arról, hogy a vizsgált rendszer vagy koponens milyen hibát hordozhat magában. Ezt célzott tesztfejlesztéssel kísérelhetik meg detektálni.
- State transition testing: Lásd Rendszerteszt 1.a példa!
- Use case testing: a készülő szoftver tervezett tipikus felhasználási módjainak szimulálása különösen a rendszerteszt és az átvételi teszt szintjein lehet nagyszerű módszer a tesztesetek megalkotására, hiszen minden termék elsősorban a felhasználónak készül, és a neki nyújtott élmény dönti el a sikerességét. Az ily módon fejlesztett tesztek olyan forgatókönyveket igyekeznek reprodukálni, ami a szoftverrel később jó eséllyel megtörténik, és az így szerzett tapasztalatokat naplózzák.
- User story testing: a user story különösen az agilis szoftverfejlesztési módszereket alkalmazó cégeknél alkalmazott formája a tulajdonképpeni rendszerkövetelményeknek. Lényege a szoftver egyes funkcióinak az olvasmányos, érzékletes ismertetése, a végfelhasználó szemszögéből. Az ezek alapján írt tesztek gyakorlatilag a use case testing dokumentáció-alapú megvalósításai.
- Domain analysis: bármilyen termék minőségének megítélése során fontos a felhasználók mellett annak a területnek (a „versenypályának") az ismerete, ahová az elkészülte után kerül. Ebből belátható, hogy megfelelő vizsgálati szempontokat alkalmazó tesztesetek megalkotásához igen hasznos tud lenni, ha a tesztelő ismeri a szektort, ahová az általa ellenőrzött szoftver kerül, legyen az akár az autó-, akár a játékipar.

Gray Box megközelítés

Gyakran célszerű egyesíteni a White Box és a Black box megközelítéseket, hogy olyan megoldásokat kapjunk, amik úgy fedik le tesztesetekkel a szoftver funkcionalitását, hogy közben annak szerkezeti felépítését is figyelembe veszik. Ezáltal a teszt által úgy növelhető a megrendelő megelégedése,hogy közben a technikai színvonal is fejlődik, valamint a tesztek az általuk feltárt hibák okának meghatározásában is segítséget nyújthatnak a fejlesztőknek.

Az itt ismertetett tesztfejlesztési technikák természetesen általában nem önmagukban állnak, és egyáltalán nem konkurensei egymásnak. A megfelelő, lehető legkevesebb idő alatt a lehető legtöbb hibát feltáró minőségellenőrzési eljárások megalkotásához célszerű belőlük minél többet felhasználni, hogy a vállalt szabványoknak, a megrendelőnek és a végfelhasználónak is megfelelő szoftvert szállíthassunk olyan költségekkel, ami a fejlesztő cégnek is vállalható.

A szoftvertesztelés életciklusa

Természetesen, mint minden munkafolyamatnak, a szoftvertesztelésnek is megvan a maga életciklusa. Ezt – az adott vállalati, céges szervezethez adaptálva – minden körülmények között érdemes szem előtt tartani, mert az így megszervezett munka hatékonyságát bizonyítja megannyi sikeres projekt, szektortól, régiótól, szervezeti mérettől függetlenül.

Ebben a fejezetben röviden bemutatásra kerül egy szoftver teljes tesztelése, a projekt indulásától annak lezárásáig.

1. A követelmények analízise

   Mint a korábbi fejezetekben többször kihangsúlyozásra került, a tesztelés ideális esetben nem a szoftver megérkeztekor kezdődik, hanem amint erre lehetőség van. Ez a gyakorlatban általában a vevői követelmények, vagy az abból létrehozott belső specifikációk elkészültét jelenti.

   Az ebben a dokumentációban rejlő hibák, ellentmondások, félreértések feltárása kulcsfontosságú, és nagyságrendekkel kisebb költségvonzattal jár, mint a későbbi fázisok során.

   Az itt alkalmazott technikák tulajdonképpen a review-k valamelyik fajtájához tartoznak. A tesztelők bevonása a követelmények létrehozásába ezen kívül a továbbiak miatt előnyös:

- Szembesülés a minőségellenőrzés során megoldandó feladatok méretével. Ez a tesztmenedzsmentet segíti, hogy a megfelelő erőforrásokat allokálja a projekt teljesítéséhez.
- Az élő diskurzus segíti a követelmények érthetőbbé tételét, valamint lehetőséget biztosít a tesztelőknek is a termék megismerésére.
- Ezen a ponton meghatározhatóak azok a tesztelési technikák és teszttípusok, amiket az adott projekten célszerű alkalmazni.
- Az egyes követelményekhez ezen a ponton már lehetséges prioritást rendelni, ami szamárvezetőként szolgálhat a későbbi tevékenységek során.
- Időben elég információt nyújt a tesztelőknek, hogy hol érdemes automatizálni a teszteket, és hol célszerűbb manuális teszteket alkalmazni. Ennek ismeretében elkezdődhet a tesztkörnyezet előkészítése.

2. Teszt tervezés

   A követelmények analízisét a tesztelői tevékenység megtervezése követi. Ennek során a tesztmenedzser elkészíti az ún. Test Plant (teszt tervet)[43] és annak alkalmazását a tesztstratégia[44] alapján.

   Emellett a jövőbeli feladatok nagyságának ismeretében meghatározza a kiszállítás teljesítéséhez szükséges (emberi és infrastrukturális) erőforrásokat, amiket a vállalat lehetőségeihez mérten igyekszik előteremteni. Amennyiben a rendelkezésre álló erőforrások mennyisége nem elégséges, jelzi azt a cégvezetés felé, hogy a tényleges tesztelői tevékenység kezdetéig ezeket a hiányosságokat orvosolhassák.

---

[43] Test Plan: az adott szoftver kiszállításának teszt oldalról történő teljesítésének a tervezete, ami listázza a fedendő követelményeket, a rendelkezésre álló erőforrásokat, a tesztelés végrehajtására delegált csapatot, valamint az egyes határidőket. Létezik egy magasabb szintű terv is a projekt teljesítésére, az ún. Master Test Plan, ami a tesztelői feladatok végrehajtása során alkalmazott megközelítések rögzítése. Leírja, melyek a termék legfőbb kockázatok a szoftver minőségét tekintve, ezek kivédését a tesztelői oldalon, valamint, hogy mely teszt-típusok kerülnek végrehajtásra, és mik a feltételei a tesztelői tevékenység lezárásának.

[44] Tesztstratégia: vállalati szintű dokumentum, mely a tesztelői szintek és a minőségellenőrzés során alkalmazott megközelítések definícóit tartalmazza.

A rendelkezésre álló tesztelő csapat között ekkor történik meg a szerepkörök kiosztása, amit a kollégák egyéni kompetenciáinak, igényeinek és egymással való kompatibilitásának figyelembevételével, akár együtt határoznak meg.
Itt azonosítják a tesztelés során alkalmazott eszközöket (tool-ok), valamint a teszt-framework-öt érintő szükséges fejlesztéseket, ezek beszerzésére illetve megalkotására külön határidőt definiálnak.

3. Teszteset fejlesztés
   Amikor a Test Plan elkészült, és rendelkezésre áll a szükséges erőforrás, elkezdődhet a tényleges tesztesetek kifejlesztése a különböző szinteken. Ezek alapját a követelmények adják, a tesztesetek létrehozásakor A szoftverteszt fejlesztés megközelítései fejezetben ismertetett módszerek megfelelő kombinációi vannak a mérnökök segítségére.
   Amennyiben szükséges a tesztek automatizálása, ezen a ponton kezdik el megírni a szükséges test scripteket[45]
   Fontos a tesztek dokumentációja, ami lehetővé teszi azok igény szerinti későbbi reprodukálását. Ezt úgy kell megvalósítani, hogy tartalmazza a tesztelőket érintő belső, technikai információkat, de a külső olvasók számára is érthető leírás is benne legyen, hogy a fejlesztők, a menedzsment és az auditorok számára is világos legyen a tesztesetek működése.

4. Tesztkörnyezet beállítás
   Bár ebben a felsorolásban ez a fázis a tesztesetek megalkotása után szerepel, sok esetben már azzal egy időben, vagy azt megelőzően elkezdik a tesztkörnyezet beállítását.
   Ez a következőket foglalja magában:
- A tesztelésnél alkalmazott hardverek és szoftverek telepítése és beüzemelése.
- A még nem létező eszközök, funkciók kifejlesztése és validációja.
- A tesztelendő szoftver telepítése és beüzemelése.
- A tesztelendő szoftver alapfunkcióinak kipróbálása egy ún. smoke test által, hogy elkerüljék az esetlegesen működésképtelen terméken, feleslegesen elégetett mérnökórákat.
- A környezet validációja, hogy meggyőződjenek róla, a tesztek során mért adatok megbízhatóak.

5. Tesztvégrehajtás
   Miután a tesztkörnyezet, az elvégzendő tesztesetek, valamint a vizsgálandó szoftver rendelkezésre áll, elindulhat annak tényleges ellenőrzése. Ebben az esetben a követelmények analízise során meghatározott prioritások szerint sorra veszik a specifikáció egyes sorait, majd végrehajtják az azok helyes működését vizsgáló teszteseteket.
   A tesztmérnök (vagy az automatizált teszt-framework) a rendszer eközbeni viselkedéséről jegyzőkönyvet készít, aminek jól látható módon rá kell mutatnia a vizsgált rendszer-követelményeknek való megfelelésére, de még inkább nem-megfelelésére.

---

[45] *Test script: az automatizált teszt-framework számára írt kód, vagy kódrészlet, ami az adott teszteset során végrehajtandó környezeti változásokat tartalmazza.*

Amennyiben egy teszt megbukik, a követelmények és a teszt, valamint (amennyiben ismert) a rendszer működésének összevetésével meg kell róla győződni, hogy a teszt által mutatott találat ténylegesen hiba-e, vagy a követelmények félreértésének az eredménye[46]. Ha ezzel együtt is megállapítást nyer, hogy az a találat tényleg hiba, a tesztmérnök ezt adminisztrálja és egy hibajegy feladásával jelzi a projekt felé[47]. Közben a már végrehajtott tesztek eredményei alapján, vagy további tesztek során az időközben verifikációra visszaérkezett hibajegyek javítását is ellenőrzik.

Amikor kész vannak a tesztek végrehajtásával, azok eredményeit, az időközben elkészült, vagy módosított teszteredményeket, valamint az adott projekten aktív hibajegyeket dokumentálják, hogy szükség esetén ezen információk a projekt rendelkezésére álljanak.

6. A tesztelés zárása

Miután a tesztek végrehajtása és dokumentációja befejeződött, elkezdődhet a tesztelői tevékenység zárása. Ennek során ellenőrzik a Test Plan teljesítését, valamint a követelmények teszttel lefedettségét és az időközben keletkezett dokumentációt archiválják.

Egy ún. retrospektív értekezleten a teszt csapat megbeszéli, mi volt az, ami jól sikerült a kiszállítás végrehajtása közben, és mi volt az, amire legközelebb jobban oda kell figyelni. Ezek alapján egy lista készül azon tennivalókból, amikkel utóbbiakat sikerülhet javítani.

+1. A tesztkontroll

Egy projekt teszteléséért felelős menedzser feladata nem ér véget a terv és stratégia megalkotásával, a szerepkörök kiosztásával. Egy kiszállítás egész életciklusa alatt fontos a munkatársakkal folytatott intenzív, de nem nyomasztó kommunikáció, aminek során időben értesülhet a felmerülő akadályokról, nem várt fejleményekről (pl. egy követelmény-csoportban a vártnál több a kielemzendő és adminisztrálandó találat), amik következtében a megadott erőforrásokkal nem biztos, hogy a feladatok a megadott határidőre teljesíthetők.

Erre a projekt sikeressége szempontjából elengedhetetlen, hogy mindig a lehető leggyorsabban reagáljon, lehetőségei függvényében külső (pl. fejlesztői) segítség igénybevételével, plusz erőforrások bevonásával, az ellenőrizendő követelmények számosságának csökkentésével, vagy a határidő újratárgyalásával. Amennyiben a menedzser ily módon rajta tartja a kezét a projektjén, az e felmerülő nehézségek ellenére is sikeres lehet.

---

[46]Az ilyen, tévesen hibát jelző tesztet fals pozitívnak hívják.

[47]A hibajegy sorsáról ezután általában a projekt menedzsmentje, vagy egy szakértőkből álló bizottság dönt, megállapítva, ki a legalkalmasabb a kijavítására. Amint a hibát úgy vélik, megjavították, visszakerül a tesztre, hogy a sértett követelmény újravizsgálatával megerősítsék a javítás sikerességét.

Automatizált tesztelés

Mikor érdemes automatizálni?

Amikor egy új szoftver minőségellenőrzése feladatként felmerül, magával vonja a kérdést, hogy manuális, vagy automatizált módon validálják-e azt.

Előbbi megoldás vitathatatlan előnye, hogy nem jelent plusz költséget az automatizált teszt-framework megalkotása (ami összetettebb rendszerek esetében egy külön termékként fogható fel), vagy megvásárlása és testreszabása (mert léteznek direkt erre a célra alkotott szoftverrendszerek is). Amennyiben azonban bizonyos szempontok a várható feladatra vonatkozóan teljesülnek, érdemes lehet átgondolni, gazdaságos lehet-e a tesztek automatizálása.

Ezt számos tényező befolyásolja, ezek közül álljon itt néhány:

- Várható a jövőben hasonló jellegű projekt?
  Amennyiben erre a kérdésre igen a válasz, úgy feltételezhető, hogy hasonló jellegű tesztek ismételt elvégzésére lesz szükség többször is. Mivel az ember természetéből adódóan nehezen viseli az ismétlődést, és nagy a kockázata, hogy nem sikerül egy tesztesetet egymás után akár csak néhányzor is pontosan ugyanúgy végrehajtania, érdemes lehet a teszteket automatizálni, hogy kivédjük az emberi tényezőt és időt takarítsunk meg a kifejlesztése után már jóval gyorsabb, automata teszttel.
- Hányszor kell lefuttatni egy adott tesztet?
  Elképzelhető, hogy bár nem várunk a jövőben olyan projektet, ami hasonló terméket állít elő, de a fejlesztés több iterációban valósítja meg a szoftvert, vagy néhány mélyebben megbúvó hibát csak egy teszt többszörös elvégzésével lehet észrevenni. Az előző pontban ismertetett emberi tényező kivédése, és az idő megtakarítása itt is érvényes.
- Mennyire kritikus az időzítés az egyes teszteseteknél?
  Az informatika világában az idő másként működik, mint az embereknél, és elképzelhető, hogy néhány milli- vagy mikroszekundum kérdése lehet a bemenetek és kimenetek változása között egy hiba megtalálása. Ezt egy ember csak erre a célra készített, programozható eszközökkel tudja elérni, ami tulajdonképpen az automatizált teszt-framework, de legalábbis annak egy kezdetleges változata.
- Milyen szabványok kötelezik a termék fejlesztését?
  Előfordulhat, hogy a fejlesztő cég által előállított szoftver olyan szabványoknak kell, megfeljen, amik kötelezővé teszik az automatizált tesztelés bevezetését, a tesztesetek precíz reprodukálhatósága miatt. Ebben a vállalatnak nincs választása, vagy automatizált tesztelést használ, vagy visszaadja a projektet.
- Milyen gyakran változik a tesztelendő szoftver felhasználói felülete/a teszt számára kritikus értékek elérése?
  Az automatizált teszt akkor térül meg, ha nem igényelnek a kódok gyakori karbantartást. Ha a fenti kérdésre a válasz az, hogy rendszeresen, úgy nem éri meg automatizálni, hiszen minden egyes változás a framework frissítését vonja maga után, ami lehet, hogy épp annyi idő, mint a tesztet kézzel végrehajtani.

- Mennyi idő áll az automatizálás végrehajtására?
  Egy tesztkörnyezet automatizálása nem kis feladat, sem akkor, ha a vállalat a saját rendszerét fejleszti erre a célra, sem, ha külső beszállítótól rendel kész megoldást (hiszen azt is be kell vezetni a saját rendszerébe, valamint a teszteket meg kell tanítani használni). Ha szűkös határidők szorítják a projektet, nem szabad az automatizálásra fordítani azt az időt, ami alatt akár tesztelni is lehetne.

- Milyen komplexitású az ellenőrizendő szoftver?
  Az egyszerűbb alkalmazások tesztelése nem igényel automatizált teszt-frameworköt, ám egy bonyolult, pl. kiterjedt adatbázist kezelő, banki szoftver átfogó vizsgálata könnyebben megoldható lehet egy gyorsan dolgozó, automatizált teszt-rendszerrel.
- Az automatizálást fontolgató cégnél megfelelő szakértelem áll rendelkezésre ehhez?
  A manuális tesztek is komoly kompetenciát igényelhetnek adott esetben, de egy automatizált teszt-framework-öt külső beszállítótól beüzemelni, vagy netán házon belül megalkotni sokkal speciálisabb, a programozás területén kiterjedt szaktudást igényelnek. Ez nem biztos, hogy jelen van minden vállalatnál, így csak akkor érdemes a tesztek automatizálásába belemenni, ha ennek megléte kétséget kizáróan igazolt.

Egy automatizált teszt felépítése

Bár nagyon sokféle teszt típus létezik, attól függően, mit, kinek, és mely szinten ellenőriznek, az automatizált tesztekre néhány jellemző általánosságban igaz lehet.

A framework használói az egyes tesztesetek meghatározásait tartalmazó decision table alapján tesztvektorokat készítenek. Ezek tartalmazzák a teszteset során használt bemenetek, és a kimenetek elvárt értékeit is. Ezek a tesztvektorok tulajdonképpen mátrixok, amik az emberi olvasó számára is átláthatók, és a gép által is feldolgozhatóak.

TV#	eletkor	szabalyzatElvogadva	kovetkezoOldal
1	19	IGEN	main/home
2	19	NEM	main/denied
3	17	IGEN	main/denied
4	17	NEM	main/denied
5	-1	IGEN	main/denied
6	-1	NEM	main/denied
7	121	IGEN	main/denied
8	121	NEM	main/denied

Ezt a tesztesetek dokumentációjában definiált lépések, mérési utasítások egy, a számítógép számára is értelmezhető kódolása követi, ami alapján a framework be tudja állítani a környezeti változókat (a bemeneteket) a tesztvektorban meghatározott értékére. Ez általában valamilyen script nyelven történik, de léteznek vizuális környezetek is erre, aminek használatához nem szükséges programozói tudás. Az így létrehozott ún. test script utasítások sorozata a teszt framework számára, ami alapján az végre tudja hajtani a szóban forgó teszteseteket.

Ha mindez kész van, a tesztvektorok végrehajtása következik, amiről a test script futása közben egy ún. tesztlog keletkezik.

Amennyiben a munkát jól végezték (ezt nem ritkán egy review során állapítja meg a készítő és egy tapasztalt kollégája), a tesztlog alapján jelenthetőek az így talált hibák, az ellenőrzés során felhasznált anyagok (a tesztleírás, az abból készült script, a tesztesetek definíciói, az azok alapján készült vektorok) pedig későbbi felhasználás céljából archiválásra kerülnek.

Szemléltető példa egy automatizált tesztre:
Egy felnőtt tartalmat szolgáltató weboldal beléptetőrendszerének működéséhez a következő követelmények tartoznak:

1. Csak 18. életévét betöltött felhasználó használhatja az oldalt.
2. Az beléptetés a következő életkorokat fogadja el: [0;120]
3. Csak az a felhasználó tekintheti meg az oldal tartalmát, aki a felhasználói szabályzatot elfogadta.

Ehhez a következő végrehajtandó lépéseket definiálta a teszt szerzője:

1. Életkor módosítása a teszteset során használtra.
2. Felhasználói szabályzat elfogadását jelző checkbox teszteset során használt értékkel való feltöltése.
3. Ellenőrzés és kiértékelés, hogy a beléptető rendszer a weboldal homepage-ére, vagy az elutasítást jelző landing site-ra vitte-e tovább a felhasználót.

A bemenetek és kimenetek lehetséges kombinációit az előző ábrán látható tesztvektorban határozták meg.

A test script pedig a fenti lépések alapján a következő, pszeudokódban:

#A bemenetek kinyerése a tesztvektorból:
eletkor = vektorbolKiolvas('eletkor')
szabalyzatElfogadva = vektorbolKiolvas('szabalyzatElvogadva')
elvartOldal = vektorbolKiolvas('kovetkezoOldal')

#Ellenőrzés, hogy a megfelelő oldalon hajtódik-e végre a teszt:
ellenoriz(belepoOldal)

#A bemenetek beállítása a tesztvektor adatai alapján:
beallitEletkor(eletkor)
beallitSzabalyzatElfogadva(szabalyzatElfogadva)

#Továbblépés az oldalról:
elfogad()

#A következő oldal ellenőrzése:
ellenoriz(aktualisOldal, elvartOldal)

Az ebből keletkező tesztlog pedig a következőképpen néz ki:

TV#	Teszteset	Mért érték	Elvárt érték
TV1	Eletkor: 19; Szabalyzat elfogadva: IGEN		
	Kovetkezo oldal	main/home	main/home
TV2	Eletkor: 19; Szabalyzat elfogadva: NEM		
	Kovetkezo oldal	main/denied	main/denied
TV3	Eletkor: 17; Szabalyzat elfogadva: IGEN		
	Kovetkezo oldal	main/denied	main/denied
TV4	Eletkor: 17; Szabalyzat elfogadva: NEM		
	Kovetkezo oldal	main/denied	main/denied
TV5	Eletkor: -1; Szabalyzat elfogadva: IGEN		
	Kovetkezo oldal	main/home	main/denied
TV6	Eletkor: -1; Szabalyzat elfogadva: NEM		
	Kovetkezo oldal	main/denied	main/denied
TV7	Eletkor: 121; Szabalyzat elfogadva: IGEN		
	Kovetkezo oldal	main/home	main/denied
T8	Eletkor: 121; Szabalyzat elfogadva: NEM		
	Kovetkezo oldal	main/denied	main/denied

Teszt-keretrendszerek

A fejlesztett szoftverek és a céges környezetek különbözősége az automatizált teszteléshez használt frameworkökben is megmutatkozik. Részletes bemutatásuk egy külön kiadvány témája is lehetne, e jegyzet célja azonban egy átfogó kép megalkotása az olvasóban, hogy néhány példa alapján fogalmat alkothasson az ilyen keretrendszerek mibenlétéről.

Az automatizált tesztframeworkök célja az előre definiált tesztesetek gépi elvégzése, miközben a mért eredményeket egy generált tesztlog formájában rögzíti. Lehetővé teszi, hogy a mérnökök csak a tesztesetek megalkotására koncentráljanak, ezáltal megtakarítva azt az időt, amit a tényleges teszt végrehajtására kellene fordítaniuk[48]. Az így felszabadított emberi erőforrás további tesztesetek megalkotására képes, ezáltal növelve a követelményeknek való megfelelésről nyújtott kép felbontását, az ún. konfidenciaszintet a szoftver minőségét illetően.

Bár a teszt-frameworkök szektorról szektorra, vállalatról vállalatra változnak, általánosságban elmondható róluk, hogy feladatuk alapvetően a környezeti változók értékeinek beállítása úgy, hogy közben a kimenetek értékeinek változását figyelik és kiértékelik.

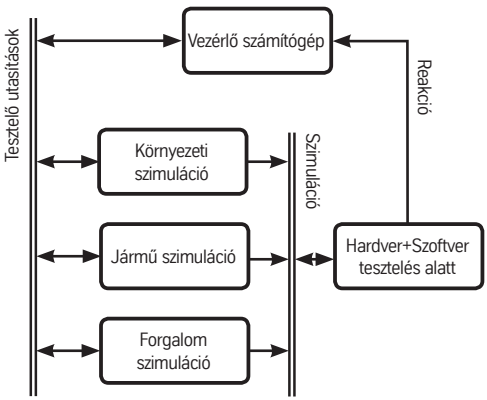
Példák automatizált teszt keretrendszerre:

1. Selenium: kifejezetten webes alkalmazások tesztelésére kifejlesztett tesztelő tool. Lehetőséget biztosít arra, hogy a tesztelők script nyelvek (pl. Python) ismerete nélkül is automatizált teszteket fejlesszenek azáltal, hogy felveszi az egér mozgását, és a közben történő aktivitásokat, majd később ezt utánozni tudja. Ezen felül elterjedt nyelvekken (C#, Ruby, Scala, Java, Groovy, PHP, Perl, Python) történő programozhatóságot is biztosít. Ily módon az előre megírt tesztprogram utasításait megkapva hajtja végre a webes felületen történő állapotmódosításokat.

[48] *Ez természetesen csak abban az esetben igaz, ha az előző fejezet szempontjai szerint indokolt az automatizálás. Egyéb helyzetben előfordulhat, hogy a teszt végrehajtása emberi kéz által rövidebb ideig tart, mint az automatizált környezet kifejlesztése.*

Szinte teljesen platformfüggetlen (Windows, Linux és macOS rendzereken is fut, a legtöbb modern böngészőben). Open-source szoftver, amit a fejlesztők letölthetnek és díjmentesen használhatnak.

2. HIL (Hardware-in-the-Loop) szimulátor: Beágyazott rendszerek tesztelésére külön, erre a célra kifejlesztett célhardver szükséges. Ez az elektronikai vezérlőre kapcsolódva azokat a jeleket szimulálja, amiket a kontroller a kész rendszerbe beépülve is kapni fog. Ezek lehetnek különböző kapcsolók, szenzorok állapotait szimuláló vonalak, vagy valamilyen, a teszt hardveren emulált adatbusz (pl. SPI, CAN, FlexRay). Az ily módon zárt rendszerbe került vezérlővel a szimuláció pontosságának, valósághűségének függvényében reprodukálható a valós használatban történő viselkedés, úgy, hogy ehhez még a végleges rendszer megépítésére sincsen szükség. A HIL-ek valamilyen módon mindig programozhatók, valamint a készítőik mérési funkciókat is integráltak beléjük, így egy ilyen tesztautomata tulajdonképpen komplett mérőrendszerként is felfogható. Bár áruk igen magas, több, hasonló profilú termék fejlesztése esetén gyorsan megtérül a rájuk való beruházás, mert jelentősen leegyszerűsítik a tesztelési folyamatot, sok esetben kiváltva a kész rendszerbe integrált vezérlő végtesztjeit.



3. Robotium: Android alkalmazások tesztelésére alkalmas open-source teszt-framework, ami a Selenium-hoz hasonlóan képes interakcióba lépni a tesztelt alkalmazással a test script által megadott utasításoknak, és a tesztesetekben megadott paramétereknek megfelelően.

Felhasznált Irodalom

BINDER, ROBERT V.,Testing Object-Oriented Systems: Models, Patterns, and Tools, Addison Wesley (1999)
HAMMER, M. & CHAMPY, J., Reengineering in the Corporation: A Manifesto for Business Revolution, New York (2001)
HOOKWAY, B., Interface, „Chapter 1: The Subject of the Interface", MIT Press (2014), p1–58.
TABUCHI, HIROKO & JENSEN, CHRISTOPHER, Now the Air Bags Are Faulty, Too, The New York Times (June 25, 2014)
ISTQB EXAM CERTIFICATION, ISTQB Exam Certification, (2015)
MARTYN A OULD & CHARLES UNWIN (szerk.), Testing in Software Development, BCS (1986), p71.
MICSKEI ZOLTÁN, MAJZIK ISTVÁN: A szofver tesztelés alapjai, *https://inf.mit.bme.hu/sites/default/files/materials/category/kateg%C3%B3ria/oktat%C3%A1s/msc-t%C3%A1rgyak/szoftverellen%C5%91rz%C3%A9si-technik%C3%A1k/12/SZET-2012_EA06_teszteles_alapjai.pdf*
PHILIP CROSBY, Developer Of the Zero-Defects Concept, The New York Times, (2001)
THE PRODUCTIVITY PRESS DEVELOPEMENT TEAM, Just-in-Time, Kaizen Pro Kft. (2011)

http://agilemanifesto.org/iso/hu/manifesto.html
http://istqbexamcertification.com
http://blog.qatestlab.com/2016/03/07/control-flow-testing/
http://softwaretestingfundamentals.com/integration-testing/
http://qatestlab.com/resources/knowledge-center/system-integration-testing/
https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2012/ee330950(v=vs.110)
www.parasoft.com/unit-testing-best-practices
https://www.istqb.org/downloads/summary/10-advanced-level-syllabus-2012/54-advanced-level-syllabus-2012-test-manager.html
http://www.professionalqa.com/test-driver
http://blog.qatestlab.com/2016/03/07/control-flow-testing/
https://www.tutorialspoint.com/software_testing_dictionary/branch_testing.htm

Jegyzetek

engineering. tomorrow. together.

thyssenkrupp