A projektek során az ismeretek hiánya, az időnyomás és a nem megfelelő szervezés generálja a legtöbb hibát, ezáltal a legtöbb időveszteséget.

3.5. A szoftverfejlesztés/tesztelés körülményei

A szoftverfejlesztés és a tesztelés igen komoly szellemi tevékenység, ami jó szellemi és fizikai állapotot kíván meg a projekt résztvevőitől.

A projekt tervezés során oda kell figyelni, hogy megfelelő körülmények biztosítottak legyenek a csapat számára.

E mellett fontos, hogy a csapat számára egyértelműek legyenek a célok és a feladatot olyan részekre bontsuk, amelyek önmagukban egyszerűek, könnyen értelmezhetőek és megvalósíthatók.

3.6. Tesztelés szerepe

A tesztelés feladata az, hogy a szoftver használata során fellépő hibák előfordulását csökkentse, a szoftver megbízhatóságát növelje és a szabványoknak, előírásoknak való megfelelőségét biztosítsa. **Tehát az ügyfél elégedettségét növelje!**

3.7. Minőségirányítás, Minőségbiztosítás, Minőség

3.7.1. Minőségirányítás

A **minőségirányítás** egy általános fogalom: Az ISO 8402 (1986) megfogalmazása szerint a minőségirányítás "a teljes körű irányítás azon komponense, amely meghatározza, illetve megvalósítja a minőséggel kapcsolatos alapelveket ... magában foglalja a stratégiai tervezést, az erőforrásokkal való gazdálkodást, valamint más rendszeres tevékenységeket, mint amilyen a minőséggel kapcsolatos tervezés, működtetés és értékelés" (ISO, 2014)

3.7.2. Minőségbiztosítás

A minőségirányításnak az a része, amely a bizalomkeltés megteremtésére összpontosít arra vonatkozóan, hogy a minőségi követelmények teljesülni fognak. (MSZ EN ISO 9000:2005, 3.2.11. pont).

A **minőségbiztosítás** önmagában is egy rendszer, amely számos komponenst fog össze:

- fejlesztési szabványok, szabályzatok kialakítása
- képzés
- tesztelés
- hibaelemzés

A jelen tananyagban csak a szoftvertesztéléssel foglalkozunk.

3.7.3. Minőség

A **minőség** definíciója: Az a szint, amikor egy komponens, rendszer vagy folyamat megfelel a meghatározott követelményeknek és/vagy a felhasználó/ügyfél elvárásainak. [4]

(http://www.scribd.com/doc/101838061/IEEE-610-Standard-Computer-Dictionary, https://cours.etsmtl.ca/mgl800/private/Normes/ieee/intro%20standards%20IEEE.pdf)

A minőség önmagában nehezen definiálható, hiszen függ a körülményektől, a felhasználótól, a lehetőségektől, sőt a kortól is, amiben vizsgáljuk.

Nehéz pontos leírást adni egy jó minőségű termékre, mégis viszonylag könnyen felismerjük a számunkra minőségi termékeket, főleg ha ezt össze tudjuk hasonlítani egy olyannal, ami nem felel meg a mi minőségi elvárásainknak. Tehát a minőség a felhasználó szempontjából szubjektív. Ez a gyártó szempontjából azonban semmiképpen nem lehet az.

Ezért ha minőségi szoftver előállítása a célunk, akkor már a tervezés szakaszában definiálnunk kell azokat a feltételeket, amelyeket teljesítve a szoftver megfelelő minőségi szintet eléri.

A minőségi szint egy olyan mértékrendszer, amely felett a rendszer elfogadható (jó) minőségű, és amely alatt nem fogadható el (nem jó) minőségű.

Mivel a határ felhasználónként és felhasználási módonként változhat, azt mindenképpen figyelembe kell venni, hogy ha a szoftverünkről az a kép alakul ki, hogy nem jó minőségű, akkor ezt a véleményt nagyon nehéz és hosszú időt igénybe vevő módon tudjuk csak megfordítani. Azonban ha tartósan megfelelő minőségű termékeket állítunk elő, úgy ha hibák merülnek fel, akkor is sokkal toleránsabb lesz a felhasználói kör.

Határozzunk meg olyan szempontokat, amelyek befolyásolják a minőséget:

- funkcionális hibák
- megbízhatóság
- használhatóság
- hatékonyság
- karbantarthatóság
- hordozhatóság

Funkcionális hibák esetén meghatározzuk azt a mennyiséget, típust és gyakoriságot, amely mellett még elfogadható minőségről beszélünk.

A három szempontot külön-külön és együttesen is vizsgálni kell.

A típus esetében, ha a hibák olyan területet érintenek, amelyek a használhatóságot nem befolyásolják, akkor még a szoftver lehet jó minőségű. Tehát ebben az esetben azt kell meghatározni, hogy milyen típusú hibák nem fordulhatnak elő a rendszerben.

Ha a hibák mennyisége – típustól függetlenül – meghalad egy bizonyos mértéket, akkor a rendszer nem megfelelő. Ez elsősorban azért van, mert eléggé általános, hogy ahol hiba van a rendszerben, ott akár olyan egyéb hibák is vannak, amelyek a tesztelés során nem derültek ki (hibafürtök). Így fel kell tételezni, hogy ott további hibák is vannak, és ha ezek száma egy meghatározott mennyiséget meghalad, akkor nem beszélhetünk megfelelő minőségről (még akkor sem, ha a hibák típusai olyanok, amelyek a használatot nem befolyásolják).

A hibák gyakorisága pedig attól függ, hogy a hibás funkció milyen gyakran van használatban. Ha a működés során az adott funkciót percenként használnánk, akkor ott nem fogadható el hiba, ha viszont egy olyan funkcióról beszélünk, amely évente egyszer használt és a következő használat is csak 10 hónap múlva lesz, addig pedig van idő azt kijavítani, akkor nyilvánvalóan ez a jelenlegi használatot és a minőséget nem befolyásoló hiba.

Megbízhatóság esetén a rendszer funkcióinak folyamatos működéséről beszélhetünk. Tehát, ha egy rendszer bár funkcionálisan nem tartalmaz hibát, de az adott környezetben folyamatosan leáll, belassul és így a funkciók nem használhatók, akkor a rendszer megbízhatatlan.

A **használhatóság (usability)** egy külön területté nőtte ki magát a szoftvertervezés területén. A usability célja, hogy a **felhasználói élmény** (**User eXperience**) minél jobb legyen. A használhatóságnál mindig azt kell vizsgálni, hogy az adott körülmények között, ahol a szoftvert használják, hogyan érhető el a maximális élmény.

Egyszerű példa, hogy ha készítünk egy weboldalt, amely számítógépen megjeleníthető és ennek elkészítjük a mobil készülékeken futó párját, teljesen mások a használhatóság szempontjai, így másképp kell működnie is annak.

Minél jobban ismerjük és vesszük figyelembe a tervezés során azokat a körülményeket, ahol a rendszert használni fogják, annál jobb lesz a rendszer használhatósága.

E mellett fontos szempont, hogy a felhasználó hogyan "éli meg" a szoftver használatát. Ha például két képernyő között – a rendszer működéséből adódóan – hosszú várakozási idő kell, akkor adhatunk a felhasználónak olyan információkat, amelyet áttekintve a képernyő váltás ideje nem tűnik olyan hosszúnak.

A **hatékonyság** esetén azt vizsgáljuk, hogy az adott szoftver hogyan gazdálkodik az erőforrásokkal. Egy adott funkcionalitást, minden tervező és programozó másként készít el. Az adott szakember előképzettsége és tapasztalata jelentősen befolyásolja az elkészített szoftver hatékonyságát.

Itt vizsgálhatjuk az adott funkciók, modulok futási sebességét, a lekérdezések sebességét és ezekre meghatározhatunk olyan mérőszámokat, amelyeket nem léphet túl az adott funkció. Például, ha van egy lekérdezés, amelyet egy perc alatt, csúcsidőben 100 alkalommal kérdezünk le, akkor meghatározhatjuk azt, hogy a lekérdezés válaszideje ne legyen hosszabb, mint 2 tized másodperc. Ehhez olyan hatékony megoldást kell választani, amely mellett biztosítható ez a lekérdezési sebesség.

Karbantarthatóság szempontjából, olyan megoldásokat kell választani, ami azt biztosítja, hogy a rendszer üzemeltetése során, könnyen áttekinthető legyen. Egyszerűen és biztonságosan elérhetők legyenek a rendszer naplói (a logok). A gyakran használt adminisztratív funkciók könnyen elérhetők legyenek.

Az üzemeltető csapat folyamatosan kapjon információt a rendszer működéséről, az esetleges hibák, amelyeket az üzemeltető nem lát, akár automatikusan bekerüljenek egy **issue tracking** (**ticketing**) rendszerbe, ahol azok a megfelelő hibakezelési folyamatot indítsák el.

Hordozhatóság szempontjából fontos, hogy a rendszer úgy épüljön fel, hogy az a környezet (hardver, szoftver) változásaira az előzetesen tervezettnek megfelelően tudjon reagálni.

A hordozhatóság szintje mindig egy fontos tervezési szempont és jelentős mértékben meghatározza a fejlesztés módját és idejét.

Ha példaként veszünk egy mobil alapú szoftvert, akkor láthatjuk, hogy jelenleg több száz különböző típusú eszköz használja az Android operációs rendszer különböző verzióit. Belátható, hogy ha csak a legelterjedtebb eszközök felbontását akarjuk kezelni a legelterjedtebb operációs rendszer verziókon nagyon komoly tervezési és tesztelési feladatok elé nézünk.

3.8. Tesztelés mennyisége és a kockázat

3.8.1. Tesztelés mennyisége

A tesztelés mennyiségének megállapítása során figyelembe kell venni azt, hogy a rendszer milyen célból készült és a szoftver egyes területeire milyen hatása lehet egy hiba bekövetkezésének. Ha ezt végiggondoljuk, akkor láthatjuk, hogy egy olyan funkciót, amelyet kevesen és nagyon ritkán használnak, valamint az ott bekövetkező hiba hatása is nagyon kicsi, nem érdemes olyan részletes tesztelésnek alávetni, mint egy gyakran használt és a folyamat szempontjából kritikus szoftverelemet.

Ezt végiggondolva láthatjuk, hogy a teszttervezés során milyen szempontokat kell figyelembe venni annak érdekében, hogy az optimális teszt mennyiségét meg tudjuk határozni.

A tervezés során meghatározunk egy olyan optimum sávot, amely költségben még elfogadható és biztosítja a megfelelő minőségi szintet.

Ez így azonban még mindig nagyon általános ezért érdemes bevezetni a kockázat fogalmát. A **kockázat** (**risk**) az a tényező, amely a jövőben negatív következményeket okozhat. Általában, mint hatás és valószínűség jelenik meg.

A **hatás** (**impact**) számítása folyamatonként elvégezhető vagy tapasztalati úton becsülhető.

Példa. Egy vállalati szoftver esetében, ha az adott folyamat nem használható 3 óráig, akkor 200 felhasználó, aki ezen a rendszeren keresztül dolgozik, nem tudja végezni a munkáját. Így a lemaradás 600 munkaóra lesz az elvégzett feladatokban. Ha ezt túlórával kell pótolni, akkor annak költsége 4 500 Ft/órával számolva 2 700 000 Ft bérköltséget jelent (az egyéb veszteségekkel még nem is számolva). Ha