ma a vészhelyzeteket azokban a szimulációs szoftverekben próbálják ki, amelyek játékokból fejlődtek valódi kiképzést segítő eszközökig.

A játékszoftverek általában mégis a szórakoztatás eszközei és elvárjuk, hogy szoftver hiba nélkül biztosítsák a maximális élményt. Ha a szoftver nem megfelelően működik, akkor az első játékosok vagy újságírók által írt tesztek negatívan tüntethetik fel azt és ez akár el is döntheti egy játék kereskedelmi sikerét.

3.4. A meghibásodás okai

A szoftver meghibásodások oka származhat a szoftver működési környezetéből és magából a szoftverből.

A környezetből származó hibák a szoftver működésére ugyanúgy hatással vannak és a legtöbb esetben a felhasználó nem érzékeli, hogy mi a valódi kiváltó ok.

3.4.1. Hardver hiba

A szoftverek minden esetben valamilyen hardver környezetben működnek, így azok problémái közvetlen hatással vannak annak működésére. Az esetek egy részében a hardver hiba az egész eszköz működésképtelenségéhez vezet, ilyen esetben a kiváltó ok könnyen felismerhető. Ha a hardver hibája olyan típusú, hogy az eszköz látszólag működőképes, akkor ez nagyon megnehezítheti a hiba felismerését és kiküszöbölését.

Hardver hiba lehet például egy olyan érzékelő meghibásodása is, amely adatokkal lát el összetettebb szoftvereket. Például, ha egy hőfok vagy nyomás érzékelő hamis adatokat ad, akkor a rendszer szoftver a hamis adatok alapján vezérli az eszközt. Ez például egy autó motorban, akár a motor részleges vagy teljes meghibásodását is okozhatja, de egy repülőgépen akár súlyosabb baleset kiváltója lehet. Természetesen ez ellen lehet védekezni az egyes rendszerek többszörözésével és az egymástól eltérő értéket adó hardver eszközök közötti prioritás meghatározásával. Ez a tervezés feladata, a tesztelés során a kritikus rendszereket úgy kell vizsgálni, hogy az ilyen esetekben való működés is egyértelműen igazolható legyen.

A hardver hiba következhet nem csak a korábban működő eszköz meghibásodásából, hanem abból is, ha az adott hardver eszköz valamilyen gyártási hiba miatt nem hozza a megfelelő teljesítmény paramétereket.

Példa: Megtörtént eset, hogy egy szoftver a terheléses tesztelés során nem hozta az elvárt paramétereket. Az egyes komponensek vizsgálata alapján nem volt oka a lassulásnak.

Kisebb terhelésnél tökéletesen működött a rendszer, egy bizonyos terhelés felett azonban exponenciálisan megnőttek a válaszidők. Csak hálózati kapacitás problémára lehetett gondolni. Külön megvizsgálva a hálózatot a szükséges kapacitás többszörösét is gond nélkül biztosította, így ezt ki lehetett zárni. Maradtak a szerverek vagy a kliensek. A kliensek régen használatban lévő eszközök voltak, minden más rendszerrel működtek megfelelően. A szerverek viszonylag újak voltak, így a figyelem ezekre irányult. A szervereket szétszedve kiderült, hogy a gyártó az összerakás során a 1 GB kapacitású hálózati kártya helyett 100 MB kapacitásút szerelt az eszközbe – rosszul felcímkézve – így ennek alacsony kapacitása okozta a problémát. (A gyártó 4 órán belül cserélte.) A vizsgálattal elveszett idő 100 munkaóra felett volt.

3.4.2. Hálózati hiba

Az egyes számítógépek együttműködését a különböző hálózatok biztosítják. Ezek lehetnek a jól ismert IP alapú hálózatok (LAN, Internet), de gondolni kell olyan hálózatokra is, amelyek valamilyen speciális számítógépeket kapcsolnak össze. A mai korszerű autókban 30-nál több különálló számítógép van. Az ezek közötti együttműködést is hálózatok, úgynevezett adatbusz rendszerek biztosítják.

Az adatok áramlása elsődleges fontosságú az egyes számítógépek között, így ha ebben hiba van, az lelassítja vagy megakadályozhatja a működést.

Amikor a hálózati hibákat kell vizsgálni, nincs könnyű helyzetben a tesztelő, hiszen a hálózatok önmagukban is sok komponensből álló rendszerek. Ráadásul itt a különböző rendszerek adatai "egy csőben" áramlanak, így egymásra is jelentős hatással vannak. A tesztelés tervezésénél és a rendszerek vizsgálatánál ezért kell külön szerepet szánni a hálózat megfelelő vizsgálatának.

A hálózat viselkedése jelentősen változik a terhelés függvényében. A növekvő terhelés a hálózat bedugulásához is vezethet, amely akár a szoftver rendszerünk leállását is okozhatja, hiszen az egyes szoftverkomponensek nem kapnak időben választ a többi komponenstől.

Általános tapasztalat, hogy a szoftverrendszerek tervezése és tesztelése során mondjuk egy vállalati hálózatot "végtelen" kapacitásúnak tekintik. A hálózat eszközök fejlődése miatt, ma már sok lokális hálózat ténylegesen jelentős kapacitásfelesleggel működik. Amikor új rendszereket telepítünk, egyre több adatot áramoltatunk át a hálózaton, ha erre a szoftverek tervezői és tesztelői nem figyelnek oda, akár az új rendszer hozhatja el azt a telítettségi szintet, amely már problémát jelent. Természetesen ilyenkor egy hálózati eszköz hibája a terheltség miatt végzetes hibát okozhat a rendszerek működésében.

Nehezíti ennek tesztelését, hogy a tesztrendszereket általában elkülönítetten kezelik az éles rendszerektől ezért nehéz valós körülményeket előállítani a tesztelés során.

Gondoljuk bele: melyik vállalat szeretné, hogy a napi működése közben az új rendszer terheléses vizsgálatát ugyanazon a hálózaton végezzük vizsgálva, hogy esetleg nem okoz-e problémát.

3.4.3. Konfigurációs hiba

A rendszerek tervezésénél fontos szempont az újrafelhasználhatóság. Ha a rendszereket csak a szoftverkód módosításával lehetne különböző környezetekbe telepíteni, az igencsak költségessé és lassúvá tenné azok elterjedését. Ezért már nagyon korán megjelent a lehetőség a rendszerek konfigurálhatóságára. A konfiguráció segítségével – az előre meghatározott lehetőségek alapján – jelentősen befolyásolhatjuk a rendszerek működését. Ez jelentősen kitágítja a rendszer felhasználási lehetőségeit, azonban ez azt is jelenti, hogy az egyes konfigurációs beállításokhoz meg kell határozni a megfelelő működést.

Sok rendszer akár több száz paraméter segítségével konfigurálható és azok értéktartománya is több tíz elemből állhat.

Belátható, hogy ez a tesztelést nagyon komoly kihívások elé állítja. Nagyon felértékelődik az adott rendszer szakértőinek tudása. Természetesen ilyen esetekben is előfordulhat olyan konfigurációs beállítás, ami nem érvényes és a rendszer nem megfelelő működéséhez vezet.

A rendszer tervezésekor ez már látszik, így célszerű olyan önellenőrző mechanizmusokat bevezetni, amelyek vizsgálják, hogy az aktuális konfiguráció érvényes-e.

Ez viszont nem ad választ olyan esetekre, amikor az egyik rendszer érvényes konfigurációja egy másik rendszerrel való együttműködésben már nem megfelelő működést okoz.

Példa: Tipikus probléma a változó terhelés (ez számos esetben megjelenik). Egy nagyvállalati környezetben működő rendszer adatbázisa általában több rendszert szolgál ki. Ennek nyilvánvalóan erőforrás optimalizálási okai vannak. Amikor az adatbázisokat telepítik, meghatározzák azokat a paramétereket, amelyekkel optimálisan tud működni. Ilyen paraméter az engedélyezett kapcsolatok száma. Ha ennek a paraméternek a beállítása nem követi egy, a funkciók bővülésével egyre növekvő terhelésű rendszer igényeit, az előbb-utóbb azt eredményezi, hogy "elfogynak" a rendelkezésre álló adatbázis kapcsolatok és az alkalmazás nem tud kapcsolatot felépíteni. Ez egy üzleti szoftver esetében

általában egyenlő azzal, hogy működésképtelenné válik. Ebben az esetben is fontos felhívni a figyelmet arra, hogy a tesztelés során törekszünk arra, hogy az "éles" környezethez a leginkább hasonló működési módon teszteljünk ez azonban legtöbbször a különböző – egyébként egymástól független – rendszerek egymásra hatásai miatt nehezen kivitelezhető.

3.4.4. Kezelői hiba

Azoknál a rendszereknél ahol a működésnek szerves része az emberi beavatkozás, ott természetesen az emberi hibát is számításba kell venni.

Ezekre a tervezésnél is érdemes felkészülni és a felhasználó tevékenységét "vezetni" kell. Ha egy meghatározott vállalati folyamatról beszélünk, akkor természetesen általában jól vezethető a felhasználó. Be kell építeni megfelelő érvényesítési eljárásokat az egyes beviteli mezőkhöz és számos olyan eszköz van, amellyel csökkenthető az ilyen hibalehetőség. Kizárni azonban ezeket sajnos nem lehet.

Példa: Egy vállalati rendszerben az ügyfél megkeresése a személyes adatai alapján (név és születési idő) történik. A felhasználónak azonban nagyon bonyolult a neve és a telefonos beszélgetés alapján a kezelő rosszul rögzíti azt a rendszerben. Ráadásul a születési idejét is elgépeli. Mind a kettő érvényes adat a rendszer szempontjából, azonban az ügyfelet nem lehet a keresési eljárás segítségével megtalálni a rendszerben. Ilyenkor érkezik egy hibajelzés, hogy eltűnt az ügyfél. Természetesen a vizsgálat megállapítja, hogy a rendszer megfelelően működik, de sajnos a felhasználó számára ez komoly problémát jelent. A tervezés során olyan megoldásokat kell választani, amelyik csökkentik a kezelői hibák valószínűségét és hatását.

3.4.5. Nem megfelelő környezet

Ahogy a konfigurációs fejezetben is említettük, a rendszereknek számos különböző környezetben kell működnie.

A tervezés során meghatározásra kerül az a számítógépes környezet, ahol a rendszernek működnie kell. Amennyiben a környezet nem olyan, mint amire a rendszert tervezték, nem garantálható a rendszer megfelelő működése. A környezet alatt értjük a szoftverkomponenseket (operációs rendszer, web böngésző, adatbázis-kezelő stb.), a szükséges hardver elemeket (szerverek, kliens), a hálózatot is.

Manapság gyakori példa az Internet világában, hogy egy elkészített szoftver meghatározott böngésző típus(okkal) és azok verzióival kompatibilis, azonban az új verziókkal vagy más típusú böngészőkkel nem, így amikor ezekkel használjuk, az alkalmazás nem működik megfelelően.

Nagyon fontos, hogy amikor egy rendszer elkészül, akkor meghatározásra kerüljön az a környezet, amelyben a megfelelő működés biztosított.

Ezért amikor egy hibajelentés készül minden esetben nagyon fontos, hogy az tartalmazza, hogy milyen környezetben állt elő a hiba, ezzel rengeteg többletmunka elkerülhető.

Környezet alatt érthetjük viszont azt a fizikai környezetet is, amelyben ennek a rendszernek működnie kell.

Gondoljunk bele, hogy egy repülőgép, amelyik felszáll egy sivatagi országban plusz 45 fokban, néhány perc alatt felemelkedik 12 000 méter magasra és ott akár mínusz 60 fok is lehet, de ezzel együtt a légnyomás is lecsökken így a nyomásértékek alapján működő szoftvereknek is figyelembe kell venniük ezt és a más módon kell működniük. Ez olyan extrém terhelésnek teszi ki a hardver eszközöket is, amelyet azoknak el kell viselniük.

3.4.6. Adatsérülés

A szoftver tervezésekor meghatározásra kerül, hogy milyen input és output adatokkal működik megfelelően.

A rendszer működése során eltárol bizonyos adatokat a memóriában vagy az adatbázisban. Ez a rendszer szempontjából zárt, tehát a rendszer nem "feltételezi", hogy ezek megváltozhatnak. Azonban ha valamilyen ok miatt az adott struktúrában letárolt adatok megsérülnek (HDD vagy memória hiba) és a rendszer erre nincs felkészítve, akkor ez a rendszer nem megfelelő működéséhez vezet.

Ha az adatok struktúrája sérül meg, akkor az felismerhető, és erre megfelelő hibakezelési eljárás építhető.

A hibák azonban lehetnek olyanok, amely nem érintik az adatok struktúráját, tehát a rendszer fel tudja azokat dolgozni (pl. egy numerikus érték esetén, ha a hibás érték még belefér a lehetséges értéktartományba). Ekkor a rendszer nem is érzékeli a nem megfelelő működést, de az output eredmények már hibásak lesznek. Ha az adatsérülés okozta nem megfelelő működés nagy kockázattal jár, akkor célszerű adatellenőrző algoritmusok használata, amely az adatok alapján különböző ellenőrző

számokat generálnak, így ha az adatfeldolgozás során az ellenőrző szám nem egyezik a korábban eltárolt értékkel, akkor feltételezhető az adatsérülés és ennek kezelésére megfelelő eljárás felépíthető.

3.4.7. A szoftver készítése során indukált hibák

A szoftverek meghibásodása származhat a külső körülményeken túl olyan okokból, amely az elkészítéskor elkövetett hibákra vezethetők vissza.

3.4.7.1. Emberi tévedés, elírás

Az emberi tévedés az egyik legkomolyabb hiba ok. Ez jöhet az igény megfogalmazásakor elkövetett hibákból. Például a tervezés során kimarad valami vagy rosszul definiálunk adatokat, algoritmusokat, amelyek később az üzleti folyamat végrehajtása során okoznak problémát.

3.4.7.2. Ismerethiány

Egy szoftver tervezésénél nagyon fontos az adott területen szerzett tapasztalat, azonban a folyamatos fejlődés miatt még ilyen esetben is előfordulhat ismerethiányból következő olyan megoldás, amely nem megfelelő működést eredményez.

Az ismerethiány lehet technológia, fejlesztéstechnikai, módszertani vagy üzleti ismerethiány.

A legtöbb esetben a technológia ismertekkel rendelkező fejlesztő csapat nem rendelkezik olyan üzleti tudással, amely elegendő a szoftver részletes és az igényeket kielégítő megtervezéséhez. Ezért a különböző területeken tudással rendelkező csapattagoknak együttműködve kell létrehozniuk a megfelelő megoldást.

Ez az egyik leggyakoribb probléma. A korszerű szoftverfejlesztési módszertanok egyik legfontosabb előnye, hogy az üzleti tudással rendelkező szakembert épít be a fejlesztő csapatba és a szoftvert is prototípusok készítésén keresztül a lehető leghamarabb megismerteti a később végfelhasználók képviselőivel.

3.4.7.3. Segédprogramok hibái

A szoftverek számos komponensből épülnek fel, ezek megfelelő együttműködése valósítja meg szoftver célját.

A mai szoftverek esetében számos olyan komponens is a rendszer része, amelyek valamilyen külső forrásból érkeznek, ezek lehetnek számos szoftverbe beépülő segédprogramok. Ezek hibája a megvalósított szoftver működését is befolyásolja. A tesztelés során külön hangsúlyt kell fektetni arra, hogy jól elkülöníthetők legyenek ezek a modulok és az esetleges hibák esetén ezek könnyen felismerhetők legyenek. Az ezekben bekövetkező hibák azonban komoly gondot okozhatnak a szoftver készítőinek, hiszen ezek "házon kívül" vannak, így a hibajavítás időben nagyon elhúzódhat, illetve bizonyos esetekben a harmadik fél ezt nem is hajlandó elvégezni.

Ezekben az esetekben megtörténhet a probléma körbefejlesztése, ami általában azt jelenti, hogy a hibás működést okozó eseteket speciális kódrészletekkel lekezelik. Ez sem fejlesztési sem tesztelési szempontból nem túl ideális, mert külön figyelmet kell rá szentelni, és ha a segédprogram fejlesztője kijavítja a hibát, akkor ez a szoftverben újabb problémákat okozhat az által, hogy a korábbi hibás működést kezelő kód a most már jól működő segédprogrammal nem a megfelelő működést produkálja.

3.4.7.4. Rohammunka és szervezési hibák

A szoftverfejlesztés általában költséges tevékenység. A projektek esetében ráadásul a piac és a technológia fejlődésével is meg kell küzdeni. 4-5 év alatt a szoftverfejlesztési technológiák gyakorlatilag kicserélődnek vagy olyan fejlődésen mennek keresztül, amely egy korábban készült rendszert teljesen elavulttá tehet.

Az üzleti szoftverek területén még az üzlet folyamatos változásával is meg kell küzdenie a szoftverfejlesztő és tesztelő csapatnak.

Ez sok esetben eredményezi azt, hogy egy projekt végrehajtására a piaci hatások miatt kevés idő áll rendelkezésre.

Ha ezt nem ismeri fel a megrendelő és a projektet végrehajtó csapat ebből komoly időnyomás alakul ki.

Ezeket természetesen nagyon nehéz kiküszöbölni ezért inkább olyan módszereket kellett kifejleszteni, ami kezeli ezeket a helyzeteket.