**BUDAPESTI GAZDASÁGI EGYETEM**
**PÉNZÜGYI ÉS SZÁMVITELI KAR**

# TERMÉK-, PROJEKT- ÉS TECHNOLÓGIAI KOCKÁZATOK MINIMALIZÁLÁSA WEBES ÜGYVITELI RENDSZEREK AGILIS MÓDSZERTAN SZERINTI FEJLESZTÉSE SORÁN

**Belső konzulens: dr. Gubán Ákos**

**Külső konzulens: Egyed Zalán**

**Csiki-Mara Viktor Tigran**
**Levelező**
**Gazdaságinformatika szak**
**Pénzintézeti informatika szakirány**

**2019.**

# NYILATKOZAT

Alulírott ***Csiki-Mara Viktor Tigran*** büntetőjogi felelősségem tudatában nyilatkozom, hogy a szakdolgozatomban foglalt tények és adatok a valóságnak megfelelnek, és az abban leírtak a saját, önálló munkám eredményei.

A szakdolgozatban felhasznált adatokat a szerzői jogvédelem figyelembevételével alkalmaztam.

Ezen szakdolgozat semmilyen része nem került felhasználásra korábban oktatási intézmény más képzésén diplomaszerzés során.

Tudomásul veszem, hogy a szakdolgozatomat az intézmény plágiumellenőrzésnek veti alá.

Budapest, 2019. év december hónap 12. nap

.....

hallgató aláírása

# *Köszönetnyilvánítás*

*Ezúton szeretnék köszönetet mondani mindazoknak, akiknek a szakmai, strukturális, morális, vagy filozófiai támogatása nélkül ez a dolgozat nem jöhetett volna létre:*

- *intézményi konzulensemnek, dr. Gubán Ákos tanszékvezető tanár úrnak, valamint Szigili Krisztina tanárnőnek;*
- *szakmai konzulensemnek, Egyed Zalánnak;*
- *jó barátomnak, Deli Sándornak;*
- *az összes tanáromnak és tanárnőmnek, akik átsegítettek a tanulmányaim folyamatán;*
- *testvéreimnek: Endrének, Attilának, Lászlónak, Tündének, Csillának és Kingának;*
- *szüleimnek, Csabának és Máriának;*
- *mindazoknak, akik a felmérésben részt vettek;*
- *és végül, de elsősorban Istennek.*

# Tartalomjegyzék

# 1. BEVEZETÉS

## 1.1. A témaválasztás indoklása

Idestova nyolcadik esztendeje foglalkozom munkaviszony keretében végzett szoftverfejlesztéssel, azon belül is webes rendszerek és alkalmazások fejlesztésével. Az eltelt időben számos projektet implementáltam úgy *front end*, mint *back end* szinten, a kiszolgálói oldalon elsősorban PHP nyelven, változatos technológiák és keretrendszerek felhasználásával.

A legtöbb esetben a *projektkockázat* minimalizálására való törekvéseink sikerrel jártak, azonban ugyanez nem mondható el a *termékkockázatról.* Bár volt példa régiós szinten több százezer egyedi érdeklődőt megmozgató megoldásokra vagy éppen nemzetközi cégcsoport a mai napig működő portálrendszerének megvalósítására is, sajnos több esetben előfordult, hogy az átadott, akár használatba vett termék nem váltotta be a hozzá fűzött reményeket, és ez vagy tartós ügyfél-oldali elégedetlenséghez, vagy a termék átadást vagy elindítást, rövidebb-hosszabb idejű működést követő teljes mellőzéséhez vezetett.

Egy fejlesztőből az ilyen jellegű visszajelzések változatos reakciókat válthatnak ki. A legegyszerűbb esetben a *nem az én gondom* gondolatkör mentén kezelhetik ezt az érintettek; ez azonban álláspontom szerint nem felel meg a *strukturális fenntarthatóság* iránti követelményeknek. Nyilvánvaló, hogy egy beosztott munkavállalótól nem követelhető meg, hogy a feladatkörén kívül eső kérdéseket firtassa; ugyanakkor véleményem szerint a *due diligence* körén belül esik, hogy a feletteseit az ilyen események eshetőleges bekövetkeztéről, amennyiben ennek valószínűségét észleli, értesítse.

Több éve érlelődik bennem a gondolat, hogy kellene fejleszteni egy olyan web alapú, moduláris, könnyen kezelhető, minden szempontból *alacsony kockázatú* ügyviteli rendszert, amely mikrogazdálkodók, kis- és középvállalatok számára egyaránt megoldást nyújthat a mindennapi ügyviteli problémáik *információ-technológiával támogatott integrációjára.*
Nemcsak az elsődleges szempont, hogy ne megbonyolítsa, hanem valóban megkönnyítse az érintettek napi ügymenetét, hanem az is, hogy minél inkább mentes legyen a *leggyakrabban tapasztalt, illetve hallott* problémák, valamint a feltárandó kockázatok többségétől.
Ilyen értelemben ez a cég szempontjából *pilot projektnek* tekinthető, a nyert tapasztalatokat jövőbeni projektekben is kamatoztatni kívánjuk.

Mivel párhuzamosan egy másik, teljesen eltérő szakirányú (technikusi) képzésben is részt vettem, a projekt csupán idén jutott el oda, hogy az infrastrukturális és a jogi háttér megfelelően rendelkezésre álljon – viszont ezen képzés alatt is bepillanthattam az informatikában kevésbé eminens emberek felhasználói elvárásaiba.
Természetesen a mai gazdasági körülmények közepette jelenleg számomra nem volt elképzelhető, hogy ezen projektet *kvázi főállásban* futtassam, emiatt a fejlesztés teljes munkaviszony *melletti* projektként zajlik.

Amint véleményem szerint ez egy legalább közepes mennyiségű időt a szoftveriparban eltöltött fejlesztőtől *szakmailag elvárható*, igyekszem figyelemmel kísérni más, szintén az ezen piaci szegmensben tevékenykedő szereplők, és fejlesztők tevékenységét, problémáit, és folyamatait. Ennek érdekében a teljesség igénye nélkül

- podcastokat, online beszélgetéseket hallgatok;
- nyílt forráskódú projektek fejlesztéséhez, javításához járulok hozzá;
- kisebb, eredeti fejlesztője által hátrahagyott nyílt forráskódú projektek karbantartását végzem leágaztatást (*forkolás)* követően;
- szakmai csoportokban veszek részt tagként vagy adminisztrátorként;
- fejlesztői találkozókra, előadásokra (*meetup)* járok;
- fejlesztői, üzemeltetői konferenciákon veszek részt;
- illetve egy ilyen fejlesztői konferencia-sorozat szervezésében magam is közreműködtem, egyebek mellett a felhívásra (*Call for Papers*) benyújtott előadási anyagok áttekintésével, véleményezésével, megvitatásával.

Mindezen tevékenységeknek köszönhetően nyílt némi rálátásom mind különféle projektek fejlesztési, mind különböző fejlesztők személyes vagy szociális problémáira, illetve kisebb-nagyobb piaci szereplők marketingjére és belső folyamataira is, és számos, általam értékesnek tekintett tapasztalatot szereztem.

Ilyen tapasztalat például, hogy a legtöbb *junior / mid-level* fejlesztő nem tud, illetve *nem akar* az ügyfél fejével gondolkodni, emiatt az ebből folyó kockázatok kezelésére nem alkalmas. Szintén hasonló, hogy itthon még mindig rengeteg helyen *rugalmasan kezelik* az architekturális tervezést (vagyis előfordul, hogy inkább rábízzák a leglelkesebbre, mintsem külön költséget generáljanak ezzel).
Ez természetesen projekt- és csapatkockázatot is generál.

## 1.2. Célkitűzéseim a dolgozat során

Jelen dolgozatban mindenek előtt arra keresem a választ, hogy a fennálló problémák feltárását követően, azokra lehetőség szerint megoldást vagy épp *work-aroundot* nyújtva, ezen túlmenően pedig az azok jelentette korlátok között lehetséges-e ma itthon egy olyan, *nem eldobható,* szolgáltatásként értékesített, elsődlegese web-alapú szoftvertermék létrehozása, amely a feltárandó problémák többségére *ténylegesen* megoldást nyújt.

Ennek érdekében az alábbi kérdésekre igyekszem elsődlegesen megtalálni a választ:

- Milyen főbb tényezők jelennek meg vonatkozó kockázatként az irodalmi, kutatási adatok szerint a hasonló projektek során?
- Mely érdekelt felek (*stakeholder-ek)* azonosíthatóak az ilyen jellegű projektekben?
- Hogyan jelenik meg a főbb érdekelt felek véleménye az előzőekben feltárt tényezőkről? Hogyan hat ez egy hasonló projekt *piacképességére*?
- Melyek azok az irodalmilag dokumentált fejlesztési *best practice*-ek, módszertanok, tervezési minták, amelyek hasznosíthatóak a projekt által?
- Ezek közül melyek a leginkább alkalmasak, illetve esetlegesen milyen módosításokat igényelnek a feltárt kockázatok csökkentése céljából?

# 2. MÓDSZERTANI ÖSSZEFOGLALÓ

## 2.1. A feldolgozandó szakirodalom forrása, előzetes következtetések

Szakirodalmi forrásként elsődlegesen a ResearchGate, a SemanticScholar, a ScholarlyCommons és az Academia.edu szolgáltatás; különböző (elsősorban egyesült államokbeli, brit és svájci) egyetemek kutatóinak nyilvánosan, az intézmény hivatalos felületéről elérhető publikációi, valamint tudományos kiadványok szolgáltak. Ezek jellemzően *peer-review*-n átesett, számos esetben iparági *best practice-é* vált publikációk. Emellett szerepelnek szemelvények tudományos konferenciák összefoglaló kiadványaiból, a szoftverfejlesztés, a csapatpszichológia, illetve a menedzsment témájában megjelent könyvek is; végezetül pedig különböző szakmai szemléletű blogok és újságcikkek. Az elsőre szokatlannak tűnő konklúziókat igyekeztem több forrásból is ellenőrizni. Több helyen utaltam is erre: vagy azért, mert például három forrás is alapvetően egyetértett valamely kérdésben, vagy éppen arra volt szükséges rámutatnom, hogy két forrás eltérő aspektusát emelte ki esetleg ugyanannak a jelenségnek.

Az irodalom előzetes áttekintése alatt a kockázatok főbb csoportjait sikerült azonosítanom; ezekkel kapcsolatban két következtetést sikerült levonnom.

Az első *előzetes konklúzió*, hogy úgy az Európai Unióban, mint hazánkban, a kis- és középvállalatok alapvetően úgy a termelés digitalizációjában, mint a – számomra érdekesebb – IT-támogatott ügyviteli integrációban *jelentős lemaradásban vannak* mind a nagyvállalatokhoz, mind az adott területen működő jogalkotók és szakmai szervezetek által optimálisnak tekintett, illetve szorgalmazott szinthez viszonyítva (DESI, 2019).

*Ezt a következtetést én a dolgozat során elfogadom, mintegy axiómaként; sem megerősíteni, sem cáfolni nem kívánom.*
Ennek oka, hogy egyrészről minden korábbi személyes tapasztalatom, és az áttekintett irodalmi adatok egyaránt ezt támasztják alá, másrészről ha mégis meg kívánnám ezt vizsgálni, az egy külön, *túlnyomórészt elméleti* értekezés témája lehetne.

A második konklúzió, hogy a célom nem lehet az előzetesen azonosított, potenciális ügyfeleknél megjelenő kockázatok közül sem a *gazdasági/konjunkturális*, sem például a *csapatkockázat* minimalizálása, mivel ezek a fejlesztési projekt szempontjából *externáliák*. *(Ugyanakkor például a fejlesztési módszertan megválasztásánál törekedni lehet és kell is ezek esetleges negatív hatásának minimalizálására.)*

Emiatt *alapvető fontosságú*, hogy a fejlesztést megelőzően képet kapjak arról, hogy az érintett célcsoportok véleménye szerint a két fő kockázati terület (a belső, *termék-/projektkockázatok*, és a külső, a terméktől, projekttől független kockázatok) *hatása* az érintettek IT-támogatott ügyviteli integrációval kapcsolatos jelenlegi vagy jövőbeli terveire nézve *hogyan oszlik meg*.

Mindezeken felül számos technológia, fejlesztési módszertan és tervezési minta áttekintésére került sor, ezek adott célra való megfelelőségére a kutatás második része keresi a választ.

Ezzel kapcsolatban a domináns álláspont az, hogy *a szoftverfejlesztés egy time-to-market – teljesítmény trade-off*, ahol a gyors fejlesztést és ezáltal piacra kerülést támogató technológiák jelentős része számottevő erőforrás-igény / teljesítményvesztés árán éri el – a külső szemlélő számára – ugyanazt az eredményt.

Ezt mindenképpen szükséges megvizsgálni a szakirodalomban, és egyes, főbb, később körülményesen lecserélhető alaptechnológiák esetében *teljesítménymméréssel* alátámasztani.

## 2.2. Kutatási módszertan

### 2.2.1. Kérdőív

A kutatás első része egy kérdőíves kutatás, amely az előzőekben megfogalmazott, alapvető fontosságú kockázatcsoport-megoszlást hivatott feltárni.

A kérdőív elsődleges célcsoportjai a *hazai* mikrogazdálkodók, kis- és középvállalatok keretein belül tevékenykedő *ügyvezetők/döntéshozók*, illetve *a tényleges felhasználók*, akik valamely ügyviteli rendszert a napi munkavégzés során használnak.
Ezt a kettős szegmenst azért tartom fontosnak, mert a döntéshozó bármely projekt szempontjából nyilván nem mellőzhető, ugyanakkor a tényleges felhasználó véleménye kulcsfontosságú a *termékkockázatra* gyakorolt hatása miatt.

A szelektivitás tehát magas; az esetleges non-reprezentativitásban kizárólag a kitöltők száma játszhat szerepet, az érintettség hiánya pedig alig vagy egyáltalán nem.

Hazánkban a KSH 2019. szeptemberi, legfrissebb adatai szerint a létszám alapján 1 457 484 mikro-, 33 468 kis- és 5 181 középvállalat működik (KSH [2], 2019.). *Közülük a mennyiségük, rugalmasságuk és az erőforrás-igényük miatt, valamint az elérési lehetőségeim okán is elsősorban a mikro- és kisvállalkozásokra kívánok fókuszálni.*

A célcsoportok szeleteinek elérésében néhány, az informatikai-gazdasági tudományok területén (elsősorban kockázatitőke-szektorbeli, akadémiai, online marketing ügynökségi és vállalkozási tapasztalattal rendelkező) ismerős és családtag vesz majd részt, akik saját üzletfeleik között, KKV egyesületekben, illetve a Hiventures kezdeti fázisban lévő *start up* vállalkozói körében segítik majd a kutatásom.

A megcélzott érintettek válaszadási készségét megbecsülni nem tudom. Bízom abban, hogy sikerül kellően kiterjedtre kivitelezni a kérdőívet annak érdekében, hogy az érintettek *joggal érezzék úgy*, hogy kíváncsi vagyok a véleményükre – *engem* például ez motivál a leggyakrabban valamely kérdőív önkéntes kitöltése során.
Járulékos motivációt, *jutalmazást természetesen* nem alkalmazok az adatgyűjtés során.

A minta mérete és a teljes halmaz mérete közötti szignifikáns eltérés, valamint a várható elemszám arra enged következtetni, hogy valamely statisztikai szignifikancia-

vizsgálatnál a *Likert skálán neutrális szentimentumra standardizált, így Gauss-eloszlásban* ábrázolható adatokra a z-próbát nem, viszont a t-próbát lehet alkalmazni. Ez azonban csak valamely esetleges korreláció vizsgálatánál válik szükségessé.

A kérdőív készítésénél alapverően a *Stanford Egyetem Társadalomtudományi Kutatóintézetének ajánlásai* szerint jártam el (VANNETTE, D. L., 2014); a *Prezi felhasználóiélmény-kutatási vezetőjének javaslatait* is figyelembe véve (KOVÁCS ZS., 2017).

A Stanford módszertantól, illetve Kovács javaslataitól egyetlen pontban térek el, teljesen szándékoltan. A szentimentum-vizsgálat során több ponton *elfogult* kérdéseket alkalmaztam. Ennek kifejezett célja van: segíthet felmérni, hogy ki az, aki a valódi véleményét osztja meg, és ki az, aki valamely esetleges *szervezeti elvárások* mentén válaszol.

A *kérdések* kiválasztásánál a fő cél az volt, hogy a *célcsoport témához való hozzáállását, ezen belül is az előzetesen feltételezett fenntartásainak valós okára fény derüljön*. Ezt kettős szemlélettel vélem megvalósítani: először a konkrét IT megoldásokkal kapcsolatos véleményekre kérdezek rá, majd a *külső* kockázatok egyes csoportjaira próbálok rákérdezni *indirekt* módon, úgy, hogy ezek alapján – *szükség esetén* – az esetleges *inkonkluzív (következtetés levonására nem alkalmas)* hozzáállásúak kiszűrhetőek legyenek. *Ez indokolja*, hogy a kérdések között néhány, tartalmilag a hipotézishez ugyan igen, a témához azonban csak érintőlegesen kapcsolódó kérdés is szerepel. (A kérdőív összeállításának részleteit a *II. mellékletben* foglaltam össze).

*A Stanford módszertan szerint meg kell jelölni annak valószínűségét, hogy az extra kérdésekre adandó választ hasznosítani fogom. Erre a jelenlegi válasz, hogy ez abszolút a kiértékelt adatoktól függ: szoros eredmény esetén ezen változók használhatóak az adattisztítás specificitásának beállítására, amelyet alapesetben nem tartom szükségszerűnek alkalmazni. Erre az eredmények kiértékelésénél kitérek.*
*Emellett – a módszertanban foglaltaknak megfelelően – a kérdések sorrendje sem mellékes: a szubjektív kérdések mindenképpen a kérdőív végére kell kerüljenek, hogy az általuk esetlegesen előidézett negatív tudatállapot ne befolyásolja a korábbi, objektívnek tekintett mérési adatokat.*

A kérdőív előtesztelésre kerül; ebben a tervek szerint a fent, a célcsoport elérésénél megemlített személyek vesznek majd részt.

### 2.2.2. Adatvédelem, felhasznált kérdőív-szoftvermegoldás, környezet

A kérdőív készítéséhez a nyílt forráskódú *LimeSurvey* szoftver 3-as verzióját használtam, *saját üzemeltetésű infrastruktúráról (nginx, PHP, MySQL) kiszolgálva*, a Tools for Research csoport által alkotott *skin*-el. Az erre a célra létrehozott aldomain dedikált HTTPS tanúsítványt kapott.

A tapasztalatom szerint *mind az alapszoftver, mind a skin hozzáférhető verziója küzd problémákkal és hiányosságokkal, az utóbbinál jelezve is van, hogy "work in progress";* igényelt némi időt és módosítást, hogy megbízhatóan működésre tudjam bírni.

*A mostanában tapasztalható adatvédelmi problémák miatt nem akartam szolgáltatásként értékesített, külföldről kiszolgált felületet használni erre a célra.*

Kiváltképpen a *Google Forms* lehetőségét vetettem el eleve. Ennek fő oka, hogy a célcsoportba olyanok is beletartozhatnak, akiknek ez kizáró ok, vagy éppen befolyásoló tényező lehet; emellett pedig *aránytalannak tartom*, hogy a Google-nek, lévén tudomása a bejelentkezett fiókok használatáról, a saját kérdőívemmel segítek az adatszerzésben, amely előnyt utána számomra nem biztosít.

A beállítást és a kérdőív létrehozását követően a szoftver azonban maximális stabilitás mellett működött; nagyon hasznos funkciója pedig, amire nem számítottam, hogy *a "félbehagyott" kitöltéseket is naplózza, az addig adott válaszokat tárolja*. Ez kifejezetten hasznos, ha a szubjektív kérdések valamely résztvevőt a kitöltés megszakítására ösztönöznének.

*Elméletileg* a megjelenés teljesen *reszponzív,* minden operációs rendszerrel és HTML5-kompatibilis böngészővel működik, és semmiféle extra bővítményt vagy egyebet nem igényel; a *gyakorlatban* viszont előfordulhat, hogy a kérdésmátrix megjelenítése a karakterek nagy száma miatt *a tableteknél kisebb kijelzőn torzulást szenved*.

Viszont a fentiek miatt a kevésbé elszántaknak a LimeSurvey alkotói által hostolt, üzemeltetési nehézségekkel nem járó példányt javaslom hasonló célra.

Ezzel együtt is bizonyos, hogy az okostelefonnal, vagy internet-hozzáféréssel rendelkezők közül *senki sem* eshet el a kérdőív kitöltési lehetőségétől; a kitöltés a *korszerűtlen* eszközöktől az egészen modernig mindenen biztosított.

Fontos kitérnem rá, hogy a kérdőív és a megvalósítás *teljesen GDPR-konform:* az egyetlen egyedi adat, amit gyűjtöttem, az IP cím, amely nem kapcsolódik az adott kitöltésekhez; kizárólag az esetleges visszaélések szűrésére szolgál; *semmilyen harmadik fél részére továbbításra nem kerül, sem analitikai, sem egyéb célból;* továbbá – amennyiben nem észlelek visszaélésszerű kitöltést – 3 nap elteltével a naplófájlokkal együtt automatikusan törlődik; adatkezelési tájékoztató és szabályzat készítésére és közzétételére vagy elfogadására emiatt *szükség nincsen.*

A nyert adatok kiértékeléséhez az *IBM SPSS 23*-as verzióját, és a LibreOffice Calc táblázatkezelőjét használom. Néhány ábra eredetije külső forrásból származik (ezeknél ezt kifejezetten jelöltem), és átszerkesztésük – elsősorban a szöveg nyelve miatt – általam történt; egy másik része *az SPSS-ben készült, és GIMP-el került feliratozásra,* egy harmadik részükhöz az *EdrawMax* program linuxos verzióját használtam, végül a többi ábra *HTML, CSS és JS "nyelven", ad hoc* jelleggel került megvalósításra, a *Firefox* böngésző saját *Gecko HTML* motorja által előállítva.

### 2.2.2. Fejlesztési technológiák, tervezési minták használhatóságának és teljesítményének az összehasonlítása

A szoftveriparban számos tervezési minta és módszertan terjedt el az utóbbi évtizedek során kisebb-nagyobb mértékben; ezek elterjedése mögött azonban eltérő okok sorakoznak.

A tapasztalataim nyomán a fejlesztés *agilis módszertan* szerint fog zajlani; tehát *funkcionális specifikációk nem*, kizárólag *interfész-specifikációk* jelennek majd meg, az előbbek helyét pedig átveszik a *user story-k*.

Eltérés az *agilis kiáltvány 1. pontjától*, hogy mivel tapasztalatom szerint a kényszeres sietség rengeteg problémát szül úgy a tervezésben, mint a gyakorlati megvalósításban, emellett pedig a projektben előzetesen nem jelenik meg külső ügyfél, valamint a rendelkezésre álló idő is korlátozott, így a *folyamatos* szállításra hangsúlyt kívánunk fektetni, a *mielőbbire* azonban – előzetesen – kevésbé.

Az egyes konkrét technológiák tágabb, a célra való alkalmasság szerinti kiválasztását az irodalom elemzését követően, az ott nyert adatokból fogom elvégezni, míg ezekből a *ténylegesen felhasználandó* részhalmaz, akár konkrét implementációk szintjén *teljesítménymérés (benchmarking) módszerrel* kerül megvalósításra.

A teljesítménymérés során terveim szerint a Microsoft (MEIER, J. D. ET AL., 2007.) és a Compuware (BARBER, S., MASON, C., 2011.) által alkalmazott *best practice*-eket fogom alapul venni.

A már most látható eltérés annak köszönhető, hogy az említett módszertanok alapvetően *iteratív*, vagyis egyetlen, rendelkezésre álló, már meglévő termék *teljesítményének optimalizálására*, egymást követő mérésekhez készült, ezért ott nyilván előzetes kritérium az *elvárt teljesítmény mérőszámainak (ún. "performance acceptance" kritériumok)* meghatározása.

Én azonban *komparatív* teljesítmény-elemzést szándékszom végezni, ráadásul technológiákon, nem kész alkalmazásokon, vagyis ilyen adatok nyilván nem állnak még rendelkezésre (ahogyan azok előállításának létjogosultsága is megkérdőjelezhető volna ezen ponton).

# 3. WEBES ÜGYVITELI RENDSZEREK FEJLESZTÉSÉBEN ÉS BEVEZETÉSÉBEN AZONOSÍTHATÓ KOCKÁZATOK

## 3.1. Az IT-támogatott ügyviteli integráció helyzete és kockázatai hazánkban és azon kívül

### 3.1.1. A modern társadalom kihívásai

A mára már elterjedt *modern társadalom* megnevezés sok, több esetben számottevően eltérő értelmezést tükröz. Avgerou például Touraine koncepciójával helyezi kontrasztba a "modern" és a "hagyományos" társadalmakat, amely elvei szerint a társadalmi életkörülmények pusztán a logikus gondolkodással javíthatóak. (CH. AVGEROU, 2000.). Ezt követően közgazdászként Weber formálisracionalitás-elméletét állítja ezzel párhuzamba, mintegy egyenlőségjelet állítva Touraine érvelési módszertana ("reason") és a racionalitás közé. Ezt követően rávilágít, hogy a weberi racionalitás-elméletnek vannak kritikusai, ami álláspontom szerint abszolút indokolt: egy ilyen azonosítás az érvelési metodikát kizárólag a *homo economicus* öncélúságával társítana össze, amely közgazdasági szempontból egyesek számára az egyetlen racionalitás, miközben például a társadalmi felelősségvállalás *jótékonysági tevékenység* (FRIEDMAN, 1970.).

Én a saját megközelítésem során ragaszkodom a *logikus érvelés* és az *öncélú racionalitás* közötti megkülönböztetés fenntartásához, és elzárkóznék az utóbbi Friedman-i értelmezésétől, már csak azért is, mert álláspontom szerint a haladás valóban indokolt részének is gátját képzi a közgazdasági racionalitás-értelmezés társadalom- és közösségellenes hatása.

### 3.1.2. A globalizáció és a verseny kockázatai

Már a 2000-es évek elején, a globalizáció hajnalán rámutattak, hogy minden egyéb folyamatot változatlannak tekintve a globalizáció az úgynevezett méretgazdaságosság irányába mutat, azaz általában véve kedvezőtlen a kisebb cégekre nézve. Ennek oka egyebek mellett az, hogy a nagyobb, több piacon operáló szereplők könnyebben kiegyenlíthették az egyes részpiacokon elszenvedett veszteségeiket máshol generált nyereséggel. (MUNDIM, A. ET AL., 2000.). Emellett az sem elhanyagolható, hogy az információ-technológia akkori elterjedése mellett a költségek kizárták ezen cégeket az integrált rendszerek fejlesztéséből vagy bevezetéséből.

Mundim és társai a nagyvállalatokkal való versengés helyett a nemzetközi ellátási láncba való bekapcsolódást, valamint úgynevezett "virtuális hálózatokba" való tömörülést jelzi. Mint az utókor bebizonyította, a nem *niche* szegmensben tevékenykedő kkv-knak valóban ez a két módszere maradt a túlélésre. Az előbbire kiváló példa hazánk autóipari beszállítói szegmense, míg az utóbbira a világszerte elterjedt internetes *affiliate (jutalékos rendszerű)* programok.
Valójában ezek egyike sem kecsegtet fényes lehetőségekkel.

A beszállítói láncba való berendezkedés rendkívül magas áttétes ágazati kockázatot, illetve kitettséget hordoz magában, miközben jelentősen korlátozza az elérhető hasznot egyebek mellett a magas beszállítói versenyeztetés; ezt némileg csak az egyes ágazatokban előforduló tanúsítási kényszer szorítja valamelyest ellenőrzött keretek közé.
Az *affiliate* struktúra pedig nyilvános, látványos, és azonnal *szemet szúr* a konkurrenciának: a nyilvánosan elérhető magas jutalék ígérete árulkodik arról, hogy az adott termék vagy szolgáltatás magas profitrátával értékesíthető, így kimagasló prioritást ér el a másolási, hamisítási célpontok listáján. Talán ennek is köszönhető, hogy napjainkban az ilyen tevékenység szinte kizárólag szerzői jog alapján replikálható, vagyis elhanyagolható változó költséggel létrehozható műpéldányok, valamint élőmunka-igénytől mentes szolgáltatások; emellett pedig *vitatható hasznosságú vagy marketing-etikájú* termékek értékesítésére korlátozódik.

Emellett az sem elhanyagolható szempont, hogy a verseny – amely a közgazdaságban alapvető fontosságúként van kezelve – helyzete sem egyértelmű. Már a fentiekből kitűnik, hogy valójában a kis- és középvállalkozások *nem* egyenlő feltételekkel versenyeznek.

Egyfelől a fentiek miatt a kitettségük jóval magasabb, nemcsak a méretgazdaságosság miatt. Az Allianz, az érintettek reprezentatív mintájának megkérdezésével készült 2018-as kockázati barométere szerint 33%-kal a legmagasabb szintű a kockázatok listáján az üzletmenet meghiúsulása (beleértve az ellátási lánc megszakadását); ez hat százalékpontos emelkedés a megelőző évi adathoz képest (Allianz Risk Barometer, 2018). *(Érdekesség, hogy a második helyen a kiberbűnözés és információ-technológiai incidensek végeztek; erről később érdemes lesz még szót ejteni.)*
Szintén jelentősen növeli a verseny jelentette kockázatot úgy a kiszervezés, mint a nyersanyag- és/vagy készletbeszerzés során megvalósuló vásárlóerő-alapú árelőny hiánya.

*Hazánkban a helyzet, ha lehet így mondani, még elkeserítőbb.*

A KSH által közölt 2015-ös adatok szerint a vizsgált 23 EU tagállamban a 250 főnél kevesebbet foglalkoztató vállalkozásokra vonatkoztatva a hazai vállalkozások átlagos nettó árbevétele 295.9 ezer EUR, amely Szlovákia, Csehország, Bulgária és Litvánia előtt az ötödik legalacsonyabb érték, míg a hozzáadott érték 55.1 ezer EUR, ami Szlovákiát, Bulgáriát és Csehországot előzi meg (KSH[1], 2016).

Ráadásul ez vélhetően a közeljövőben nem fog változni, az adatok abba az irányba mutatnak ugyanis, hogy a vállalkozói szféra összetétele nem fog számottevően módosulni a közeljövőben.

Holló szerint elsősorban a hazai vállalkozói kultúra helyzete kifogásolható. (HOLLÓ E., 2017.) Ennek okaként mutat rá (egyebek mellett) a vállalkozói szellemre irányuló médiafigyelemnek; a vállalkozás, mint kívánatos pálya népszerűsítésének; valamint a

vállalkozói ismeretek általános iskolában történő oktatásának uniós átlagtól jelentősen elmaradó szintjére.

Én ezzel szemben kiemelném, hogy az Eurostat 2006-os adatai szerint Magyarországon a népesség 57.6%-a nem beszél idegen nyelven (Eurostat, 2016.). Ez az adat egyébként így is jelentős előrelépés a 2010-es 74.8%-os arányhoz képest. Tapasztalataim szerint a szakiskolákban, OKJ-s képzésekben van vállalkozásiismeret-oktatás a gyakorlatban is (hogy ebből mi marad meg, az más kérdés). A fenti kockázatok tükrében, illetve azt tekintve, hogy hány korábban sikertelen, jelentős adósságot felhalmozott (és esetleg azóta is törlesztő) vállalkozóval találkoztam életemben, nem tartom különösképpen előremutatónak a vállalkozóvá válásra való ösztönzést általános iskolában.

A fentieken túl van egy másik, kettős vetületű kockázat is: a munkaerő-hiány és fluktuáció. Poór kutatása során a válaszadók első helyen jelölték meg a fizikai munkaköröket, mint az iparban legnehezebben betölthető állásokat (POÓR J. ET AL., 2018). A magánszférában általánosságban az informatikusok és karbantartók jelentek meg legnehezebben pótolhatóként.
Ez egy vállalkozó számára kettős hatás: egyfelől számíthat arra, hogy piacképes tudás esetén rendkívül gyorsan el tud helyezkedni munkavállalóként; emellett viszont ha munkaerő-igényes tevékenységet folytat, számolnia kell azzal, hogy egy esetleges bővülés, vagy munkavállalói felmondás esetén nem fog tudni új munkaerőt tervezhető határidőn belül, gazdaságosan felvenni.

Ez álláspontom szerint nemcsak a vállalkozások beindításának szempontjából jelent szignifikáns rizikót, de a már viszonylag stabilan és tervezhetően működő vállalkozások gazdaságilag esetleg indokolt bővülését is igen alapos megfontolás-kényszer hatálya alá helyezi.

### 3.1.3. Az információ-technológiai megoldások terjedése és terjesztése

Egy három német szövetségi államban (Észak-Rajna-Vesztfália, Bajorország, Baden-Württemberg) 1400 cég bevonásával elvégzett elemzés a cégen belüli részlegek, illetve a partnereik közötti digitális kapcsolódások, valamint az okoseszközök használati arányát vizsgálta a cégméret függvényében, emellett pedig felkérte a vizsgált cégeket saját maguk digitális fejlettségének értékelésére (SCHRÖDER, CH., 2018).
Az elemzés adatai szerint a kisvállalkozások számottevően felülértékelték a saját digitalizációjuk fokát. A kutatás a két legfőbb problémaként a digitális stratégia hiányát, és szervezeti problémákat emelte ki.

A szervezeti problémák közül a legjelentősebb a szervezeti ellenállás a változással szemben. Ennek okai sokfélék lehetnek. Ma nyilvánvalóan a leggyakoribb ok az ellenállásra bármilyen digitális változással szemben az érintett aggodalma, hogy a saját vagy valamely kollégája munkája feleslegessé válik. Ez egy szignifikáns és egyre erősödő kockázat. Stouten és társai munkájában a közös vízió elérése és kommunikálása aggregát módon, több forrás által megerősítve megjelenik, mint a szervezeti ellenállás leküzdéséhez *nélkülözhetetlen* lépés (STOUTEN, J. ET AL., 2018). Létfontosságúnak

tekintik azonban, hogy ez a vízió *nyílt* és *transzparens* legyen, valamint *kellőképpen absztrakt.* Ez utóbbi kritériumot például nem teljesíti, ha például egy kész helyzet elé állítják a szervezetet, hogy mondjuk holnaptól ezt és ezt az *enterprise content management* rendszert kötelesek használni, hiszen ilyenkor nincs lehetőségük érdemben beleszólni a vízió kialakításába, így nem fogják azt érzékelni, hogy a véleményük fontos vagy jelentékeny. Emellett idézik Kottert (1996), aki rámutat az üzenetben foglalt inkonzisztenciák (ilyen, ha az üzenet az elbocsátás szükségességét tárgyalja, miközben a vezetőség például fényűző nyaralásokra repül például vállalati kisgéppel) racionálisan elidegenítő hatására (esetlegesen a személyesen nem érintettek felé is).

Nem kevésbé jelent problémát az sem, amikor nem a technológiától való félelem, hanem a *régi, jól beváltnak* tekintett *módszerek* átalakítása, megváltoztatása idézi elő a *szervezeti ellenállást*.

A Lidl-nél 2011-ben döntöttek a SAP Retail rendszernek a HANA *in-memory database* alapú bevezetéséről a korábbi, belső fejlesztésű *Wawi* kódnevű projekt helyett. Az új, SAP alapú projekt az *eLWIS* kódnevet kapta. A projekt 2018-ban történő leállítása a német Computerwoche című magazinban *"eLWIS Hana alapon halott"* címmel, címlapon jelent meg (SCHAFFRATH, T., 2018). Schaffrath ugyan nem taglalja a kudarc okait, pusztán kihangsúlyozza, hogy egy ilyen tőkeerős céggel ellentétben egy *kis- és középvállalkozás* egy ilyen kudarcba biztosan belerokkan. Krabbenborg azonban onnan számítja a problémák kezdetét, hogy a Lidl menedzsmentje rájött: az SAP terméke a készletértékelést az eladási árakon végzi, miközben a Lidl Wawi rendszere a beszerzési árakkal számol (KRABBENBORG, G., 2018). A pontos részletek nem ismertek, ezért valamely felsorolt módszertan esetleges számviteli problematikájában nem tudok elmélyülni, a szerző viszont kiemeli, hogy a Lidl úgy döntött, hogy *nem fogja megváltoztatni* úgy a *mindset*-jét, mint a használt eszközeit; ezzel az elhatározással ehelyett nekiláttak az SAP-termék *testreszabásának*; továbbá felteszi egyebek mellett az általam legfontosabbnak tekintett kérdést: *hogy nem vette ezt az eltérést észre senki több éven keresztül?* Észrevételeinek legfontosabb tanulsága, hogy *a csoportosként kezelt felelősség a legtöbbször elkerülhetetlenül elkendőzéshez vezet*, és ezen az a tény sem segített, hogy a projekt ideje alatt a cég CEO-t cserélt.

A témában az *Agiles.com* cikke érdemel még szót, amely Krabbenborg érvelésével némiképp szembehelyezkedve rámutat, hogy *a testreszabás nem tartozik az SAP erősségei közé*, továbbá, hogy a *Deutsche Post* is kudarcot vallott már az SAP bevezetésével korábban (Agiles.com, 2018).

A nap végén viszont kívülről úgy tűnhet, hogy a Lidl projektje végül *egy több, mint ötszázmillió eurós tévedés* lett, amelynek összességében *nincs felelőse. (A DHL részére bevezetni kívánt rendszer vesztesége egyébként végül 345 millió euróra rúgott.)*

Hasonló megoszlás figyelhető meg némi célirányos érdeklődéssel a közszférában is. Ameddig egy vállalat ügyfelét jellemzően nem érintik számottevő mértékben a vállalat által *belsőleg* használt információs rendszer sajátosságai, addig a közszféra digitalizációja egy új helyzetet teremt: ott a szervezeti ellenállásban részt vevők,

jellemzően szintén az állásukért aggódók ellenpólusa nem elsősorban a megtakarítani vagy éppen bővülni kívánó vállalat.

Ebben a szegmensben egy új hatás jelenik meg: az ügyintézésre *kötelezett* polgárok. Az ő szempontjukból az állami bürokrácia minden fajtája negatív externáliaként kezelhető, amelyet szabályozói okokból nem tudnak negligálni, tehát – a jogalkotó esetleges segítő szándékától elvonatkoztatva – adottnak tekinthető.
Ők egy igénnyel jelennek meg az államigazgatás felé: ahogy a mindennapokban egyre több mindent intéznek számítógép, vagy okostelefon segítségével, úgy reálisan egyre kevésbé szeretnének akár órákat utazni a megyeszékhelyre, elintézni valamit, ami számukra jellemzően egy kötelező kellemetlenség.
Ilyen módon ez az igény az államigazgatás szempontjából szintén egy negatív externália, ráadásul a monopólium miatt nem járhatnak el úgy, hogy átterelik az akár az átlagnál kellemetlenebbnek tekintett ügyfeleket a konkurrenciához – *az ugyanis nincs*.

Természetesen az ügyfelek magánszférában szokatlan száma jelentősen növeli a kockázatot a kormányzati IT beruházások sikertelenségének tekintetében.

Heeks és Bailur rámutatnak, hogy az e-kormányzati kutatások alacsony száma mellett domináns jellemző az ágazati optimizmus, amely alapvetően naivitásnak tekintendő (HEEKS, R., BAILUR, S., 2018). Az általuk áttekintett kutatások több, mint fele egyszerűen "jó dolognak" tekinti a közszférában az információ-technológiát, miközben bagatellizálják a hátrányait, és teljesen figyelmen kívül hagyják a széleskörűnek nevezett meghiúsulási költségeket. Ezt a hatást a *hype* igen magas fokának tulajdonítják, amely nem független attól, hogy a témában kiadott legtöbb ismeretterjesztést a bevezetésben érdekelt IT cégek írták. Végezetül rávilágítanak arra is, hogy a témában publikáló tapasztalt kutatók (akik idézték saját magukat) *kevesebb, mint egynegyed része* helyezkedett kifejezetten pozitív álláspontra.

Ezek a potenciális nehézségek a *stakeholder-ek*re vetítve valószínűleg sokkal jelentősebbek nyugaton, mint nálunk. A fentiekben már tárgyalt munkaerő-piaci kockázat nyomán itthon egy vállalkozó akkor sincs könnyű helyzetben, ha (morálisan esetlegesen vitatható szándékkal) ténylegesen a munkaerő-költség ilyen jellegű csökkentése céljából vezetne be valamilyen adminisztrációs rendszert, hiszen, mint erre is rövidesen kitérek, az információs rendszerek bevezetése sem nevezhető általánosságban véve sem kockázatmentesnek, és ha például erre hivatkozva felmond egy munkavállalónak (vagy éppen ő fog "idejekorán" továbbállni az első érdemi ajánlatnál), akkor korántsem garantálható, hogy talál a helyére megfelelő személyt, ha a rendszer nem váltja be a hozzá fűzött reményt, vagy egyszerűen csak csúszik a bevezetés.

### 3.1.4. Informatikai Due Diligence, avagy technológiai kockázatok

Ma az informatikai üzleti projekteknek egy elenyésző része lesz sikeres. Ebben jelentős szerepük van a *diszruptív piaci szereplőknek*, akik szinte óráról órára előállnak egy új *forradalmi* ötlettel.

Bár a startup, illetve ehhez kapcsolódóan a kockázatitőke-befektetés fogalma a köztudatba leginkább a 2010. utáni "aranykora" során ivódott be, a jelenség abszolút nem újkeletű.

Proksch egy 2000-es tanulmányt idéz, amely 35 és 55 százalék közé tette akkor a sikertelennek bizonyuló kockázatitőke-befektetések számát (ZARACHAKIS ÉS MEYER, 2000, IDÉZVE PROKSCH, D. ET AL. ÁLTAL, 2016). A hivatkozott tanulmány azonban bizonyosan 2000-es megjelenése előtt íródott, 2016-ban pedig ez az állítás már bizonyosan nem állta meg a helyét.
Mi több, az általuk hivatkozott tanulmány megjelenése idején már javában zajlott a szektor első tündöklése utáni bukása: az úgynevezett *dotcom boom* kipukkanásával az elmúlt évszázad legnagyobb pénzügyi veszteségébe torkollott.
A probléma mértékét jól jelzi, hogy míg az *1929-es nagy gazdasági válság* USA-ban *1929-ben realizált tőzsdei vesztesége* a Time magazin cikke szerint 25 millió USD (SUDDATH, C., 2008) – ez 2019-es értékre átszámolva hozzávetőlegesen 375.38 milliárd USA dollár, addig a CNN Money dotcom lufiról szóló cikke szerint 280 *internetes részvény* akkori árfolyamon egyetlen éven belül 1755 milliárd USD-t veszített értékéből (KLEINBARD, D. 2000). Ez az eredeti érték 59.53 százalékának "elégése".

A hivatkozott érték tehát az összeomlás előtti időkből származik, a szerzők mégis ezt idézik 2016-ban is, aminek számos oka lehet; ebbe nem mennék bele. Fontos ugyanakkor, hogy az Európai Bizottság már 2002-ben kiterjedten foglalkozott a kockázati finanszírozási alapok létrehozásának rendszerszintű kockázataival, de elsősorban a piactorzító hatásokra fókuszált, és a régiós kockázatitőke-alapok létrehozásának feltételéül szabta a *piaci kudarc* bizonyítását (European Commission Directorate General for Regional Policy, 2002).
Mivel az alap létrehozásakor (remélhetőleg) bizonyosan nem ismert a később támogatandó piaci szereplők listája, ezért azok kudarcát nyilván nem szabhatta feltételül, így kizárásos alapon ez vélhetően a szabadpiaci "láthatatlan kéz" régiós szintű működésképtelenségét jelentette, amely a kohéziós politika célkitűzéseinek megvalósulását veszélyeztette.
Teljes mértékben rejtély ugyanakkor előttem, hogy két évvel a bolygón valaha volt legnagyobb tőzsdei összeomlás után vajon milyen kockázatot vélt a Bizottság ezzel csökkenteni; feltételül szabta azonban, hogy egyértelmű *exit*, vagyis tőzsdére lépési stratégiával kell rendelkeznie minden támogatottnak. Ez álláspontom szerint ismét a kockázat szétterítésének eszköze, amely munkahely-védelmi és egyéb célból esetileg támogatható ugyan, de jellemzően a kisbefektetők kockázatát növeli.

A dotcom boom egyértelmű tanulsága ugyanakkor, hogy az információ-technológiai terület üzletileg a legkockázatosabbak közé tartozik. Ennek főbb okai, a teljesség igénye nélkül:

### *A befektetők nem értik a technológiát*

Ennek következményeként *érzelmi* vagy *megérzés alapon* kénytelenek dönteni arról, hogy valaminek *van-e jövője*.

Napjaink egyre absztraktabb technológiai útvesztőjében lassan napról napra röppennek fel a *buzzword-ök* – e pillanatban a *neurális háló*, a *kvantumszámítógép*, a *blockchain* a dobogósok, és a gyakorlati hasznosságukról és előnyükről megoszlanak a vélemények, abban azonban egyetértés van, hogy jóval több az ígéret mindegyik téren, mint a megvalósult és gyakorlatban demonstrálható projekt – a PricewaterhouseCoopers 1378 vezető technológiai cég CEO-jának megkérdezésével végzett felmérése is erre a következtetésre jutott (GLOGER, M. ET AL., 2019).

### *A diszrupció korát éljük*

Ma a startup szcénában lépten-nyomon tetten érhető az a jelenség, hogy valaki fog egy okostelefont, összeügyeskedik rajta egy alkalmazást, amely ugyan létező probléma megoldásának láttatja magát, valójában azonban a bevételszerzésének elengedhetetlen feltétele, hogy a "szürke zónában" mozog, vagyis úgy cselekszik, mint a különféle *compliance* szabályok területi hatályán kívül esne (legyen szó a "közösségi személyszállítás" esetében a fuvarozókra, sofőrökre vonatkozó különféle előírásokról, vagy a "közösségi szállásközvetítés" adóügyi és tartalékképzési szempontjairól).

Tehát gyakorlatilag az ebben érdekeltek közösségeinek nevezik, és forradalminak tekintenek olyan piaci szereplőket, amelyek technológiai újdonságokat alig vagy egyáltalán nem mutatnak fel, jogszabályoknak való megfelelőség és morál terén azonban számos kérdést hagynak nyitva.
*Azt én ugyan nem vitatom, hogy forradalminak forradalmi, ha valaki nem vesz tudomást a szabályozásról vagy a közteher-viselési kötelezettségről, viszont például közösségeinek nevezni egy olyan szolgáltatást, amely a lakásbérleti díjak rohamos emelkedéséhez vezetett világszerte, ahol működik, finoman szólva is aggályosnak tartom.*

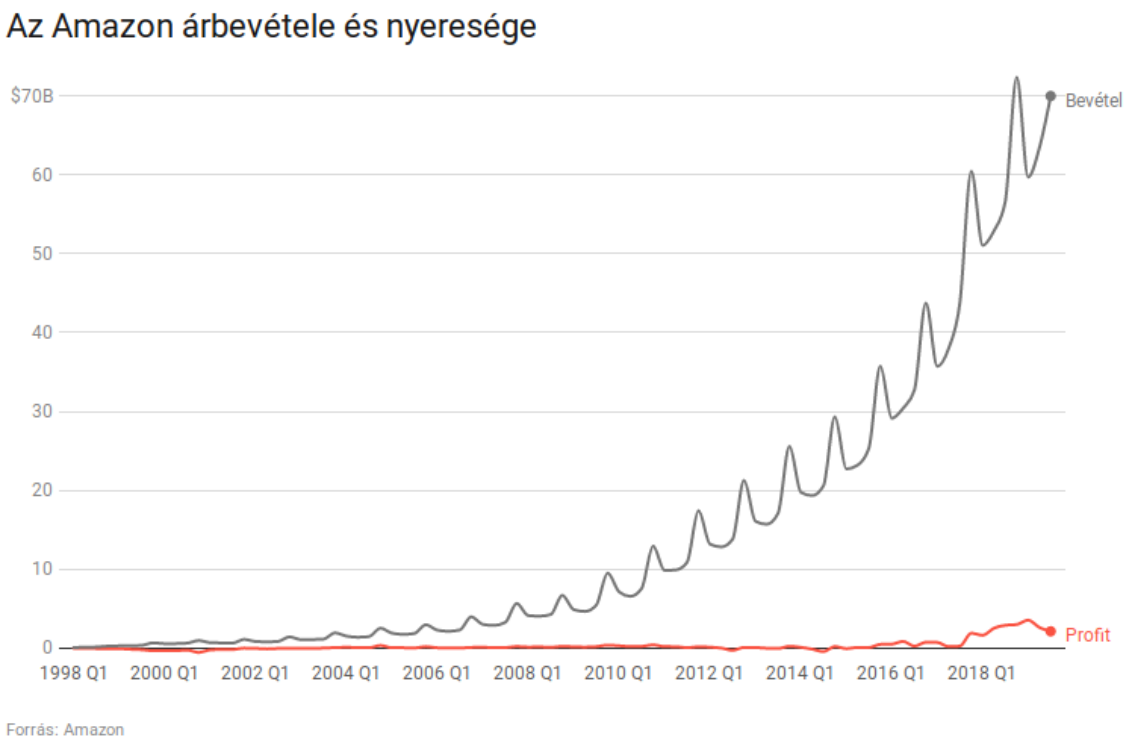
Mindezek miatt gyakorlatilag az összes, ezen szereplők esetleges pozitív bevétel-/nyereségtermelésével kapcsolatos állapotot pillanatnyinak szükséges tekinteni, amely egy adott, a társadalom által elvárt vagy akár kikényszerített szabályozási eseményt követően megváltozhat. Jellemzően nem növekményes irányba.

### *Minden ingyen van (vagy legalábbis verhetetlen áron)*

Szintén veszélyes trend, hogy érdemi anyagi és intellektuális befektetés nélkül bárki piacra léphet. Míg a dotcom boom idejében James Hong egy *low end* asztali számítógépet egy kollégiumban a benti hálózatra rácsatlakoztatva világhírű társkereső oldalt tudott indítani (PENENBERG, A. L., 2009), addig ma számos "felhő alapú" szolgáltató biztosít *ingyenes* belépő szintű csomagot (úgynevezett *freemium* modellben).

*A felhőszolgáltatások különféle kockázatairól később, a fejlesztéshez kapcsolódó elemzésnél részletesebben is szót ejtek.*

I*tt azonban mindenképpen szót érdemel az, hogy például a globálisan piacvezető Amazon Web Services felhőszolgáltató cége, az Amazon árbevételéhez képest egyre elenyészőbb mértékben termel nyereséget (1. ábra), miközben a befektetőket ez látszólag a legkevésbé sem aggasztja. Mi több: folyamatosan veszik a részvényeit, ezzel piaci kapitalizációja egyre növekszik.*




*1. ábra: Amazon árbevétele és nyeresége (1998 Q1 – 2019 Q3; mrd. USD)*
*Forrás: Vox.com*

Látható azonban, hogy ezek olyan folyamatok, amelyek nagy valószínűséggel nem tarthatóak fenn örökké. Ha pedig a korrekciós kényszer megjelenik, az az ingyenes szolgáltatások megszűnésével és az önköltségi ár alatti díjak növekedésével fog járni; ez pedig azzal, hogy az ezekkel kalkulált üzleti tervek dominószerűen borulni fognak.

De nem csak a kockázati szegmensre érdemes időt szentelni.

A korábban már említett PwC felmérés alapján a megkérdezett cégvezetők mindössze 40 százaléka véli úgy, hogy cége az elkövetkezendő évben biztosan növekedni fog tudni (GLOGER, M. ET AL., 2019).

### 3.1.5. Csapatkockázat

Az alanyok emellett kiemelik a csapatkockázatot (*team risk*): 50 százalék nyilatkozott úgy, hogy rendkívüli módon aggódik a megfelelő tehetségű munkaerő felkutatása és alkalmazása miatt, míg 55 százalék biztosnak tekinti, hogy a képességek hiánya biztosan hátráltatni fogja cégük innovációs képességeit.

*A csapatkockázat egy igen szerteágazó, és mindeközben kiemelkedően ellentmondásos terület, amire mindenképpen szükséges részletesebben kitérnem, hiszen mindezek a tényezők a tényleges fejlesztési folyamathoz, mint termelő tevékenységhez kapcsolódóan is megjelennek kockázatként, tehát a projekt szempontjából negatív externáliák lehetnek ügyféloldalról és fejlesztői oldalról egyaránt.*

A legegyszerűbb vetülete az, amelyet a cégvezetők is kiemeltek: munkaerő-hiány, megfelelő munkaerő megtalálása, "elcsábítása" a céghez; betanulási veszteség korlátozása.

Ennél sokkal az információ-technológiai ágazatban szerteágazóbb problémák vannak.
A folyamatosan jelen lévő fejlesztő-hiány és a fejlesztők feladatainak jelen idő szerinti automatizálhatatlansága *korábban soha nem látott kompromisszumokra* kényszeríti az érintett cégek döntéshozóit.

### *A kezdetek: az archetipális informatikus*

A *hagyományos* kép, amely a fejlesztőkről, rendszergazdákról az információ-technológia hőskorában kialakult, egy antiszociális, tipikusan hímnemű, durva, kapcsolatteremtésre korlátozottan alkalmas sztereotípiában manifesztálódik (TOCCI, J., 2009).
Ez a témával mélységeiben foglalkozók szerint egyebek mellett arra vezethető vissza, hogy a magasabb intellektusú egyedek már gyerekkorban kitűnnek a társaik közül, és korlátozott érdeklődést mutatnak csupán a *szocialita* viselkedésű tömegek tevékenységei iránt (UCLA CENTER FOR MENTAL HEALTH IN SCHOOLS, 2010).
Itt fontos megemlíteni, hogy ez abból a korból származik, amikor a mai *játékkonzol* fogalma teljesen ismeretlen volt, a *számítógép* pedig eleinte nem volt alkalmas játékra, később pedig a játékok rövid lefutású, fix ívű folyamatok voltak, nem ilyen *immerzívek*, mint a mai globálisan többjátékosos módban, kvázi végtelenül játszható játékok. Tehát miközben igazat adok a tanulmány szerzőinek, fontos felhívnom rá a figyelmet, hogy annak megállapításaival szemben ha *a mai generáció* nem érdeklődik a hagyományos társas tevékenységek iránt, az már nem szükségszerűen jelent kimagasló intellektust.

Napjainkra ugyanakkor a fenti személyiség szinte teljesen eltűnt.

### *Nyílt térrészű irodák*

Ahogyan hozzánk is beszivárgott a nyugati típusú nagyvállalati *team culture*, úgy hozta magával az *open plan / open space* típusú irodai elrendezéseket. Paradox módon bár ezeket a *munkahelyi kollaboráció növekedésének* ígéretével szokták népszerűsíteni, a tapasztalatok alapján ez nem is lehetne valótlanabb: az egyik cégnél lefolytatott *pilot projekt* esetében 72 százalékkal *visszaesett* a szemtől szembeni interakció a munkatársak között, miközben 56%-al több e-mailt küldtek, és 41%-al több e-mail üzenetbe kerültek be másodlagos címzettként (BERNSTEIN, E. S., TURBAN, S., 2018). A szerzők rámutattak arra, hogy az ilyen környezetben dolgozni kényszerülők próbálják magukat a lehető leginkább elszigetelni társaiktól (*jellemzően hatalmas fejhallgatók használatával*), miközben a láthatóság miatt igyekeznek a lehető legelfoglaltabbnak *tűnni*.

Ezzel kapcsolatosan szintén számottevő probléma, hogy vannak, akik a megfelelő szellemi teljesítményhez igénylik a megfelelő oxigénellátást *és* hőmérsékletet.
Cochrane The Guardian-en cikke megjelent cikke szerint például a Facebook-nál folyamatosan 15 fok van az irodában, Zuckerberg a vélelem szerint ezzel próbálja

folyamatosa éberen tartani a dolgozókat (COCHRANE, K., 2013).
Itthon a munkahelyek munkavédelmi követelményeiről szóló 3/2012. SzCsM-EüM együttes rendelet 1. mellékletében található táblázat értelmében 21-24°C közötti hőmérséklet biztosítandó irodai környezetben illetve szellemi munkavégzéshez; ez az érték a legmagasabb egyébként a táblázatban.
Véleményem szerint a 15 fok erős túlzás, a 24 viszont már sok a koncentrációhoz, ráadásul mindenhol vannak, akik 24 fokban megfáznak, tüsszögnek, mindenféle betegség tünete kezd eluralkodni rajtuk – holott valójában a 19-21 fok lenne az optimális.
Belátható, hogy ennyifajta igénynek egy nyílt légtérben nem lehet eleget tenni akkor sem, ha egyébként a klímaberendezés, a fűtés kielégítően működik. Ilyenkor a döntéshozó, de a dolgozó is választani kényszerül: vagy halad a tervekkel és cserébe "összerúgja a port" a mediterránabb vérmérsékletű kollégáival, vagy pedig beletörődik a béke érdekében abba, hogy "ez van" – ennek viszont nyilvánvalóan a tervek, a projektek látják kárát.
Sajnos tapasztalatom szerint a hazai cégvezetői kultúra sincsen azon a szinten, hogy felismerje, hogy akár egy gipszkarton- és gépészeti beruházás költsége és az általa okozott maximum két napos munkaidő-kiesés *is* bőven megtérül néhány hét, vagy legfeljebb hónap alatt, ha a kollégák ettől képesek a valódi teljesítményüket nyújtani.
Ugyanakkor a munkavállaló nem kötelezhető arra, hogy felvállaljon olyan feszültségeket, amelyek eleve nem az ő hatáskörébe esnek.

Nyilvánvaló, hogy a korabeli *informatikus-archetípusnak* ez a környezet a lehető legtöbb szempontból tekintve finoman szólva is *szuboptimális*.

### *Soft skillek*

Szintén új elemként jelent meg a palettán a *soft skillek* kérdése. A *startup* szcénában eleinte nem fektettek erre számottevő hangsúlyt. A *hagyományos* technológiai cégeknél, főleg a nyugati-globalista kultúrában (*ahol a menedzsmentben elsősorban társadalmi, közgazdasági szempontok érvényesülnek*) azonban folyamatosan növekvő követelmény, hogy a dolgozó *jól kijöjjön* kollégáival. Ugyanakkor a Lippman által idézett felmérés szerint már 2013-ban domináns volt a soft skillek hiányának a közrehatása a kvalifikált munkaerő hiányában (LIPPMAN, L. H. ET AL., 2015).
Emellett újabban mind társadalmi, mind piaci nyomás van a *startup*-okon, hogy szakítsanak az *autokratikus* jellegű vezetéssel, és tegyenek eleget mindenféle társadalmi elvárásnak. Ez társadalmilag a felhasználói bázison keresztül, míg piaci szempontból a későbbi befektetési körök vagy éppen *exit* sikerességének feltételeként lehet kikényszeríthető.

Emellett a csapatmunka mibenlétéről és minőségéről is eltérőek a vélemények. J. Richard Hackman, a "Leading Teams" című könyv szerzője például egyenesen azt állítja a Harvard Business Interview-el készített beszélgetésében, hogy *az egyik legfőbb tárgyi tévedés* a csapatmunka kapcsán az, hogy *azok a csapatok, amelyek jól működnek együtt, lesznek a legproduktívabbak*, miközben a szimfonikus zenekarokkal folytatott

vizsgálata azt mutatta, hogy a mogorva egyesületek némileg jobban teljesítettek (COUTU, D., 2009). Ennek okaként a kauzalitás elemeinek felcserélését hozza: *a sikeresen végzett munka idézi elő az elégedettséget, nem pedig fordítva*.

A másik leggyakoribb *téves közvélekedés*, hogy a nagy csapatok jobban teljesítenek, mint a kisebbek, mert például több erőforrás áll rendelkezésükre. Hackman rámutat, hogy a tagok számával közel exponenciálisan emelkedik közöttük a kapcsolatok száma, amelyeket ápolni kell.
Ökölszabályként a tíznél kevesebb fős csapatokat ajánlja, míg saját maga számára hat főben húzza meg a határt.

Ismert pszichoszociális jelenség, hogy a csapatok számának növekedésével a hatékonyságuk csökken. Ezt a jelenséget Maximilien Ringelmann már az 1890-es évektől tanulmányozta, és 1913-ban megjelent könyvének központi tételévé vált. Ezt a jelenséget róla Ringelmann-effektusnak, vagy *közös lógásnak (social loafing)* nevezték el (SCHOLTES, I. ET AL., F., 2016). Scholtes és társai ezen jelenség szoftverfejlesztői szektorra való hatását elemezték 58 nyílt forráskódú projektben 30 000 fejlesztő és 580 000 *commit* időszeletes vizsgálatával, és arra jutottak, hogy az egy fejlesztőre eső Levenshtein távolság (*gyakorlatilag a megváltozott, hozzáadott vagy törölt karakterek száma*) a projektekben regisztrált *kontribútorok* számának növekedésével csökkent – ez kvantitatív bizonyítékként értékelték a Ringelmann effektus érvényességét tekintve.
Emellett megállapították azt is, hogy a *beleszerkesztési hálózatok (co-editing networks; az a halmaz, amely az adott problémát önállóan nem oldotta meg, de a tényleges megoldás módosításába, vélelmezett "finomításába" bekapcsolódott, ilyetén a saját részvételét ténylegesen vagy hamisan növelve)* mérete lineáris fölötti mértékben növekedett. Véleményük szerint ez lehet az egyik mechanizmus a nyílt forráskódú területen való közös lógás mögött, vagyis az, hogy értelmezésem szerint létező, vagyis bejelentett problémák (*issue-k*) megoldására kevesebben vállalkoznak, mint a már elkészült megoldások "megszakértésére".

*Véleményem szerint mindezek a trendek szintén nem kedveznek a sztereotip, introverz fejlesztői személyiségnek.*

### *Diversity and Inclusion*

Emellett folyamatosan nő a legnagyobb vállalatokon a külső (és belső) nyomás az úgynevezett Diversity and Inclusion mozgalom célkitűzéseivel kapcsolatban is. Ennek a legalapvetőbb megjelenése a nők számarányának a növelése úgy a menedzsmenten, mind a fejlesztői szektorban egyaránt.

Miközben ebben a célkitűzésben aligha találhat bárki kivetnivalót, a módszertan számos kérdést felvet. Miközben a társadalomtudomány domináns mondanivalója a témában az, hogy a nemek, rasszok és egyéb védett jellemzők szerinti kvóták bevezetését egyre több szervezet és jogalkotó fontolgatja, fő érvként jellemzően a kvóta nélküli elvitathatatlan folyamat *akadozó* menetét felhozva (HE, J., KAPLAN, S., 2017), addig Maitland bemutatja, hogy a mozgalom nem akadémiai disszertációból, hanem egy legfelsőbb

bírósági véleményből származtatott, és kiemeli, hogy a szoftvermérnökök és fejlesztők piacán igen magas a verseny, ezért ott az ilyen nyomás üzletileg végzetes lehet (MAITLAND, I., 2018); Ford pedig arra helyezi a hangsúlyt, hogy ahol a D&I mozgalmat adatsorokká silányítják, ott kiveszik az üzletből a humán faktor (FORD, J., 2018).

A Google egyik Tech Talent Strategist-je a felvállaltan a menedzsment *megújítására* törekvő Management Exchange oldalon számol be arról, hogy az *extrém diverzitás* elérése érdekében *nagy árat* kell fizetni (PIGNI, D., 2018). Egyebek mellett nevesíti az *igazsággal való szembenézést*, amely (a teljesség igénye nélkül) olyan dolgokkal jár, mint a termelés megszakadása, vagy a korábban sikeres menedzserekkel szembeni új elvárás, hogy mondjanak le a csapat-alakítási jogukról egy új, akár külső szereplő javára. Azt is hozzáteszi, hogy *kevés cég képes túlélni ezt a második lépést (a diverzitási terv tényleges árának megfizetése).*

### *Kiszervezés*

Az antiszociális férfi informatikus sztereotípiáját azonban nem elsősorban az előbbiekben felsorolt jelenségek szorították napjainkra háttérbe.
A szoftverfejlesztő-hiány mára már krónikusnak tekinthető, azonban akut korában is arra kényszerítette a döntéshozókat, hogy alternatív megoldásokhoz folyamodjanak.

Első körben – elsősorban a globalizáció előnyeként számontartott költséghatékonysági szempontok által vezérelve – számos vállalatnál elkezdődött az *outsourcing*: az információ-technológiai feladatok meghatározott részét kiszervezték. Mivel az üzemeltetés kiszervezése és az adattárházak földrajzi áthelyezése energiahatékonysági, infrastrukturális, adatbiztonsági, illetve *compliance* okokból sem volt széles körben megvalósítható, így ez a változás nemzetközi téren elsősorban a fejlesztői tevékenységeket érintette.

Sajnos azonban ez számos esetben kudarchoz vezetett.

Gazdaságilag véleményem szerint az outsourcing legnagyobb kockázata a szellemi tulajdon védelme, hiszen a technológiai transzfer, a kutatási-fejlesztési költségek nélküli lemásolás tisztességtelen versenyt jelent, amelynek a hatása nem szűnik meg az adott együttműködés felbontását követően sem.
De nem elhanyagolható a kulturális különbségek jelentette probléma sem. Ugyanakkor ehhez képest jelentős az üzleti, illetve szakmai kompetencia, valamint a kommunikáción belüli transzparencia hiánya is (PHILIP, T. ET AL., 2009). Philip szerint ezek, illetve a további kiemelt faktorok még úgy is a vizsgált projektek többségének a sikertelenségéhez vezetnek, hogy a *sikertelenséget* a projekt *átadás előtti* meghiúsulásaként értelmezik, tehát nem értik bele az átadott, de üzemeltethetőségi, skálázhatósági vagy teljesítménybeli téren később, használatba vételt követően leállított projektek kudarcát.
Ez a kudarc néha *pusztán* munkahelyek elvesztésével *és* dollármilliók, milliárdok elpocsékolásával jár, máskor azonban a szereplők nem ússzák meg ilyen olcsón.
A Seattle Times cikke szerint a Boeing egy belső levélben John Hart-Smith

szerkezetmérnök már 2001-ben, három évvel a 787 Dreamliner projekt elindítása előtt felhívta a figyelmet arra, hogy számos problémát okozhat a növekvő kiszervezési kényszer (GATES, D., 2013). Később kiemelte, hogy a Boeing-nál úgy vélték, hogy a kockázatot és a felelősséget is ki tudják szervezni a feladatokkal együtt.
Tévedtek: a Dreamliner piacra lépése jelentős késéssel, hét évvel a projekt elindulása után, számos problémával – úgy szerkezeti, elektronikai, mint szoftver-oldali – került piacra, úgy, hogy a piacra lépése előtt egy évvel a tesztrepülés során hajszál híján sikerült elkerülni egy katasztrófát.
De még sejtelmük sem volt róla, hogy *mekkorát* tévedtek.

De nem szükséges a tengerentúlra menni egy kis kudarcért: Indiana állam a szociális juttatások rendszerét szervezte ki az IBM-hez és alvállalkozóihoz; az 1.4 milliárd dolláros projekt az *átadást követően* három évvel dőlt be, amikor a gazdasági helyzet miatt a segély iránti igények közel a duplájukra nőttek; ezt a terhelésváltozást egyik fél sem kívánta megfinanszírozni (Niccols, Ch., 2019). Kísértetiesen hasonló történt az IBM texasi, illetve floridai projektjével is; ebből a texasiban olyan komoly problémák léptek fel, hogy már a projekt bevezetését le kellett állítani.

A kiszervezés helyzete napjainkra se fordult pozitívba.
Mitra a HuffPost hasábjain már 2008-ban az Indiába történő IT outsourcing halálát jósolta, és 2017-ben ezt már küszöbön állóként definiálta (MITRA, S., 2017). Álláspontja szerint a mesterséges intelligencia által vezérelt automatizáció rengeteg állást fog fölöslegessé tenni. *(Habár ez kétségtelenül valós aggály, az IT outsourcing érintettségét nem sikerül egyértelműen hozzákapcsolnom.)* Emellett aláhúzza: az Egyesült Államoknak *elege van* a munkahelyek kiszervezéséből, és ezen jelenség okaként Trump-ot és a *protekcionizmust* jelöli meg elsődleges tényezőként. Véleményem szerint ez utóbbi inkább okozat, mint ok; számos okot pedig – meglévő szándék esetén – az általam jelen szakaszban bemutatottak között meg lehet találni.
Ugyanakkor *tényszerűen* jelenti, hogy az indiai IT szolgáltatás-iparág 200 000 munkahely megszüntetését *már bejelentette*.

A jelen évben eközben a Boeing legújabb, 737 MAX jelű gépéből, már a piacra kerülés után, a megrendelőknél öt hónap alatt kettő zuhant le; a balesetekben a fedélzeten lévő mind a 346 személy *életét vesztette.* N*émi, pilótahibára utaló ködösítést követően* vizsgálat indult, amely megállapította, hogy a katasztrófákat egy új, a manőverezési karakterisztikát szabályozó szoftverkomponens okozta, amely a vizsgálat jelenlegi állása szerint *hibás adatokat kapott egy szenzortól*. A vizsgálat során feltárásra került, hogy a katasztrófában szerepet játszott hibák *alapvetőek* voltak, amelyek nem függetleníthetőek attól a ténytől, hogy a gép szoftverének fejlesztését és tesztelését jelentős részben olyan, jellemzően indiai fejlesztők voltak, akik *mindössze* 9 dollárt kerestek óránként, és gyakran semmiféle repülési ismerettel nem bírtak (ROBISON, P., 2019). A 9 dolláros órabért annak tükrében szükséges vizsgálni, hogy az érintett indiai fejlesztők nem Indiában, hanem a Boeing irodája mellett, Seattle-ben, a HCL Technologies irodájában dolgoztak, jellemzően friss diplomásként. Dacára a specifikáció szerinti fejlesztésnek, szinte mindig több egyeztetésre volt szükség a

Boeing mérnökei és a HCL fejlesztői között, mert "a kód nem megfelelően volt megírva".

### *Gazdasági migráció*

Következő megoldásként a jól fizető, de betöltetlen tengerentúli állásokat külföldi jelentkezőkkel kezdték feltölteni. Bár a gazdasági migráció általános megítélése globálisan és ideológiai hovatartozás szerint eltérő, az úgynevezett *hiányszakmák* betöltését a legtöbb ország elég régóta könnyített letelepedéssel és különféle kedvezményekkel segíti.
Ez egyfelől érthető, illetve valamelyest mindig is így volt, másfelől viszont a technológiai szektorban kiemelten problémás, ugyanis a jól képzett informatikai szakemberek szinte mindenhol hiányoznak, ezért mára az informatikusok váltak a legnagyobb gazdasági migrációs csoporttá. Az Egyesült Államok Munkaügyi Statisztiaki Irodája adatai szerint ma a betöltetlen állások száma ugyan csak a hatodik a rangsorban, viszont az éves medián 103 620 USD fizetés messze az első (US Bureau of Labour Statistics[1], 2018), így a betöltetlen állások betelepülőkkel történő betöltésének gazdasági hatása a második legnagyobb a regisztrált kórházi nővérek után, ráadásul a prognosztizált 26 százalékos tízéves növekedés miatt előreláthatóan az első helyet is megszerzi (US Bureau of Labour Statistics[2], 2018).

*Mindennek a következménye az, hogy a legmagasabban kvalifikált fejlesztők, ameddig és ahol ezt megtehetik, jellemzően a jobban fizető földrajzi régiókba költöznek, ezzel az agyelszívás jelenségét fűtve.*

### *Utánpótlás, átképzés*

A jövedelmi szempontból kevésbé vonzó területeken, amilyennek nemzetközi összehasonlításban hazánk is tekinthető, ezért sokan új módszerekhez folyamodtak: megkezdődött a *szoftverfejlesztő-átképzés*. Nehéz, illetve korai lenne megítélni, hogy ez a jelenség az outsourcing-hoz viszonyítva hol helyezkedik el a káros következmények okainak rangsorában, két dolog azonban már most biztos. Az egyik, hogy ez a módszer a hírek szerint inkább szolgálja a képzést szervező piaci szereplők érdekeit, mintsem akár a jelentkezőkét, akár a munkaadókét (BOHUS P., 2019). Bár ezek a képzések előre figyelmeztetnek, hogy *junior* tudást adnak az itthon jellemzően 4-20 hónap alatt, a tapasztalatok alapján nem az a probléma, hogy sokan nem tartják be a szerződési feltételeket, és a vállalt idő előtt távoznak a biztosított munkahelyről, hanem egyre inkább az, hogy nem sikerül őket elhelyezni (ami a munkabérből levonni tervezett képzési díj esetében igencsak strukturális probémának tekinthető), és ez a trend csak nő. Ennek okát az említett cikk az iparági szereplők fokozott óvatosságával, illetve fenntartásaival magyarázza, továbbá azzal, hogy "a juniortól is munkatapasztalatot kérnek".
Joshua Weinstein, egy egyesült államokbeli technológiaikarrier-specialista azonban a kudarcért kizárólag a képzésben résztvevőket okolja (WEINSTEIN, J., 2019). Bár ez szerintem erkölcsileg mindenképpen kifogásolható, Weinstein három olyan dologra mutat rá, amelyek minden kétséget kizáróan szerepelnek a palettán.

Az egyik pontját mindenképp érdemes kiemelni: *"Kizárólag a pénzért veszel részt"*. Kifejti, hogy az emberek folyamatosan váltanak állást a magasabb fizetés reményében, és ezzel szerinte *semmi probléma nincsen*. Nyilván abszolút helyzetfüggő, hogy ezen kijelentéssel egyet lehet-e érteni, azt azonban vitathatatlanul tárgyilagosan helyezi fókuszba, hogy "*az anyagi sikerre való törekvés megtévesztően kevéssé hatékony motiváció"*.
De a másik két felvetése, amelyek az erőfeszítés, illetve az előzetes felkészültség hiányát firtatják, is igen relevánsnak tekinthetőek.
Az itthoni helyzet egyfajta specifikuma, hogy miközben Weinstein vitatható módon kizárólag a jelentkezőket hibáztatja az (ottani) kudarcért, addig itthon számos képzésszervező *kifejezetten* kisgyermekes anyukák, vagy éppen "asszisztensek, újságírók, vagy tanárok" számára hirdetik magukat, fókuszba helyezve az *átlagon felüli életszínvonalat* (ZIMRE ZS., 2018).
A formális logika általam ismert szabályai szerint *ha* azt feltételezzük, hogy Weinsteinnek igaza van, akkor megállapítható, hogy az itthon magukat ilyen jelleggel népszerűsítők effektíve *belehajszolják* a jelentkezőket a kudarcba.. *De mivel célul tűztem ki, hogy a gazdaságetika kérdéseit kizárólag annyiban érintem, amennyiben ez egy adott fejlesztés folyamatát befolyásolhatja, így nem elemezném, hogy egy kisgyermekes anyukával ez mennyire tekinthető etikusnak.*

*Azt viszont mindenképpen le kell szögezni, hogy az ezen iskolák végzőseivel szemben fenntartásokat ápoló munkaadók óvatossága mellett ez a fajta "marketingstratégia" sem abba az irányba mutat, hogy az itt végzettek jelentős része elkötelezett, szakmailag magasan kvalifikált és mindenekelőtt a további fejlődés iránt érdeklődő szakember lesz.*

### *Egyéb módszerek*

Belátható, hogy az elsődleges motiváció, mint a fizetés, bár növekszik, de nem növelhető a végtelenségig. Eközben a kvalifikált (és nemcsak a kvalifikált) szoftverfejlesztők egy számottevő része külföldön próbál szerencsét, míg hazánkat vélhetően a szektorban lassan alapvetéssé váló nyelvismeret birtokában aligha tekinti egy magát relokációra rászánó programozó célországnak.
A kereslet folyamatos bővülése mellett ez alternatív *praktikák* kidolgozására készteti a vállalatokat, amelyek célja elsősorban a más ajánlatok meghirdetői elől, esetleg már meglévő munkahelyről *átcsábítani* a megfelelőnek vélt személyt.
Ma már egy szóra érdemes programozói munkahely mellé *elengedhetetlen* az ingyen kávé és legalább szűrt víz, de a szilíciumvölgyi startup-kultúrát *másolva* számos helyen jelent meg a rekreációs helyiség, csocsóasztallal, pingponggal és effélékkel.
Ez ugyan kétségtelenül sokak számára vonzónak tűnhet, de a valóság az, hogy előre nem lehet felbecsülni ennek előnyeit, hiszen munkaidő után többnyire mindenki igyekszik haza a családjához (esetleg a törzshelyéül szolgáló bárba), az pedig csak a később megismerendő céges *hagyományokból* derül ki, hogy mennyire teszik lehetővé, hogy például a törvény biztosította nem képernyős tevékenységet a munkavállalók a csocsóasztal mellett (az ezzel járó zajt előállítva) töltsék el..

Kétségtelenül hasznos ugyanakkor a *rugalmas munkaidő*. Az, hogy a munkavállalónak nem kell szabadnapot kivennie a bürokrácia útvesztőjében történő, törvényben vagy a piacgazdaság szabályai által előírt elmélyüléshez, vagy egyéb problémái elintézéséhez, igen nyomós érv lehet egy kevésbé rugalmas ajánlattal szemben.
Divat emellett a belvárosban, *menő helyen* lévő irodát emlegetni, amely a produktivitáshoz vajmi kevéssé járul hozzá, és összességében is leginkább a biciklistáknak kedvez közlekedési szempontból – de ha valaki gépjárművel szeretne munkába járni, ilyen esetben jobban teszi, ha bérel egy P+R helyet.
Emellett vonzó tud lenni még a különféle kedvezményes adózású juttatásokat magában foglaló ajánlat is.

Véleményem szerint ennél sokkal kézenfekvőbb megoldások is szóba jöhetnek. A fentiekben taglalt nyílt légterű munkatérrel kapcsolatban megismertekkel összhangban magam például mérvadó motivációként tekintenék az open space helyetti zárható, klimatizált légterek kialakítására.

Emellett a *munka törvénykönyvéről szóló 2012. évi I. törvény (Mt.) 92.* § (4) bekezdése *egyértelműen és kifejezetten lehetővé teszi*, hogy a munkaszerződés szerinti felek az (1)-ben definiált 8 órás *teljes napi munkaidő* helyett annál kisebb, 7, vagy akár 6 órás *teljes* munkaidőben (tehát *nem részmunkaidőben*) megállapodjanak.
Sajnos azonban a a társadalombiztosítás ellátásaira és a magánnyugdíjra jogosultakról, valamint e szolgáltatások fedezetéről szóló 1997. évi LXXX. törvény *legalább heti 36 órás foglalkoztatást* ír elő, mint munkaviszony-kitételt, amely mellett például egy vállalkozó nem minősül főállásúnak.

Majdnem egy évtizede ismert már, hogy például a Google-nál a munkavállalók a rendes munkaidejük 20 százalékában *saját projekteken* dolgozhatnak (ROBINSON, A., 2018). Ez mindenhol hordoz magában kockázatot. Itthon vélhetőleg az jelentené a legfőbb aggályt (nyilván a mentalitáson felül), hogy nehéz lenne elkülöníteni a *magán* és a *központi* projektekhez fűződő adatokat, állományokat, és ekként ellenőrizni, hogy a munkavállaló mit hova tölt fel, *mit visz haza*. (Ez, úgy látszik, nem mindenhol gond egyébként.)
Ez egy teljesen más szempontból kockázatos egyébként magánál a Google-nál is, és a fejlesztésnél ennek szentelném gyakorlatilag a jelen alfejezet összes tanulságát.
Nem egyszer fordult elő, hogy a Google (illetve valamely szerkezeti egysége) *meglátta a fantáziát* egy-két ilyen projektben, és megpróbálta azt *az Internet összes felhasználója torkán letolni*.

Sajnos itthon egy rendkívül sajátos kultúra figyelhető meg: a produktivitás másodlagos, sok munkáltató tekinti központi szempontnak, hogy az előírt munkaidőt mindenki percre pontosan betartsa, miközben annak növelésére vannak törekvések. És sajnos már ezt is fejlődésnek kell tekinteni a néhány éve mérvadó "*pontban tizenhét órakor kiesik a ceruza a kezéből*" megközelítéshez képest..

***Összegzés***

Mindezen szempontok alapján véleményem szerint megállapítható, hogy a munkaadók, több valós, de vélhetően felülreprezentált szocio-politikai kényszer, valamint itt nem feltárt motivációk nyomán inkább részesítik előnyben a *status quo-t* és ezzel a beilleszkedési hajlandóságot, mint a ténylegesen meghirdetett feladatra való alkalmasságot.

Tisztán látható, hogy bármelyik utat is választja a döntéshozó, gyakorlatilag csupán a kockázatok közül választhat szabadon; kockázatmentes mozgástere a bérezéstől és a rendelkezésre álló vagy bevonható munkaerő (amennyiben van) képességeitől eltekintve *sem* igazán van.

*A vázolt helyzetnek a projekt szempontjából egyik legfontosabb következménye, hogy az ügyfélnek a projekt szempontjából megkérdőjelezhetetlen érdeke fűződik ahhoz, hogy az informatikai integrációs és fejlesztési folyamatok céljára szakmailag megfelelő munkaerőt biztosítania ne váljon szükségessé (hacsak például egy meglévő, saját fejlesztésű rendszerrel való összekapcsolás ezt nem teszi elkerülhetetlenné – ez esetben azonban már vélhetően rendelkezik ilyen munkaerővel). Ezzel ugyanis nem elhanyagolható mértékű kockázatnak tenné ki magát, amelyet itthon bizonyosan kevesen tekintenének némi "végre kevesebb Excelt kell használnunk" életérzéssel kiegyensúlyozottnak.*

*Mielőtt azonban felmerülne, hogy ettől az apróságtól eltekintve az egész alfejezet irreleváns és öncélú szempontokat vizsgált kimerítően, fontos ismét megemlítenem: van mindezeknek egy másik, ennél is sokkal komolyabb következménye, amely a fejlesztési módszertanra igen jelentős befolyással van.*
*Erre ott részletesebben kitérek.*

### 3.1.6. Termékkockázatok

Az eddig felsoroltak alapvetően *projektkockázatot* jelentenek. Nem szabad szó (pontosabban előkészületek) nélkül elmenni a *termékkockázatok* mellett sem.

A lehetséges termékkockázatokat jelen értekezés szempontjából két részre osztanám. Az első csoportba az *előre látható* termékkockázatok tartoznak, amelyek közül kettővel foglalkozom kiemelten: az *ügyféligények téves vagy hiányos felmérése*, illetve a *fejlesztési eszköztár és/vagy módszertan(ok) helytelen vagy a célt nem szolgáló megválasztása.*
Ezek megvizsgálására a 4. fejezetben, a fejlesztéshez kapcsolódó irodalom elemzésekor kerül sor.

A második csoportba az előre nem látható termékkockázatokat sorolnám, ahova olyan esetek sorolhatóak, mint például az ügyfelek esetleges módosítási igényeinek következetes és kategorikus elutasítása; az ügyfelek elvárás-/szentimentum-változása; vagy a gazdasági környezet váratlan és gyökeres megváltozása.
*Ezekre – jellegükből fakadóan – a dolgozatban nyilván legfeljebb eshetőlegesen, az előzetes elvárhatóság keretein belül tudok kitérni.*

### 3.1.7. A nyilvános felhők, mint technológiai kockázat

A felhőszolgáltatások terjedésének számos kockázata van. Ezek egy részére a kutatási eredményeknél térek ki. Van viszont egy olyan vetület, amelyre *elengedhetetlen* ezen fejezetben rávilágítani.

A felhőszolgáltatások megjelenését követően egy *új, eddig javarészt* ismeretlen technológiaikockázat-csoport *ütötte fel a fejét*: ez pedig a *hardver-elemekben* meglévő sebezhetőség. Ugyan IT biztonsági szempontból *régóta* ismertek a különböző *oldalági (side-channel)* támadások (mint például a feszültség vagy az órajel manipulálása valamely *mikrovezérlő beégetett*, védett kódjának a kiolvasására), az x86 architektúra elleni hasonló támadások még sokáig nem láttak napvilágot.

Az első ilyen probléma a Google által felfedezett és 2015. márciusában nyilvánosságra hozott *Rowhammer* kódnevű sérülékenység volt, amely a *korábbi DDR* és *DDR2* technológiákat nem, csak a *DDR3* és *DDR4* rendszerű memóriát érinti, és *lehetővé teszi*, hogy a programok *az operációs rendszer (és minden fölöttes réteg) védelmét megkerülve, egyes bitek állapotát átbillentve* adatot változtassanak meg a memóriában.

Mivel ez *hardverszintű* támadási felület, *ezért javítással teljes körűen nem küszöbölhető ki; a mai napig* az összes általánosan létező *work around-ok mindegyike* arra alapul, hogy *szándékosan lassítsák* a memória-elérést, a hiba ugyanis csak egy megadott adatelérési sebesség *fölött* jelentkezik.

Nem kellett sokáig várni a folytatásra: 2018. januárjában publikálták a graz-i egyetem kutatói a *Meltdown* kódnevű problémát, amelyet még 2017-ben fedeztek fel, és amely megfelelően kihasználva *tetszés szerinti memóriához való hozzáférést* tehet lehetővé *bármely felhasználói szinten futó program számára.* Bár az Intel közzétett az év folyamán *javításokat* (úgynevezett *mikrokód* szinten) a processzoraihoz, ezek *mindegyike teljesítményvesztést eredményezett.*

Az első hírek arról szóltak, hogy a sebezhetőségek új családja *kizárólag az Intel által gyártott processzorokat érinti*, ugyanezek a kutatók ugyanazon évben azonosították a *Spectre* kódnevű sebezhetőséget is (KOCHER, P. ET AL., 2018). EZ *minden olyan Intel, AMD, és ARM-rendszerű processzort érint, amely támogatja a spekulatív végrehajtást (tehát például szinte az összes 2006 óta gyártott Intel, 2003 óta gyártott AMD, és új generációs ARM processzort, ideértve a teljesség igénye nélkül az összes modern Apple eszköz processzorát).* Sajnos a probléma *jellegéből fakadóan* nem javítható szem szoftveres, sem *mikrokód* oldalról, és bár lehet rá *lépéseket tenni*, hogy a hatását *csökkentsék*, jelenleg egyetlen konkrét megoldást a *spekulatív végrehajtás kikapcsolása* jelentene, ami ismét erőteljes teljesítményvesztéshez vezetne (ez a gyakorlatban a többmagos architektúrát nem számítva *a 2003-2006 előtti szintre való visszatérést* jelentené). Egyetlen olyan processzorcsalád van, az AMD *Zen* architektúrájára épülő *EPYC*, amely bár szintén támadható, azonban emellett támogat úgynevezett *virtuálisgép-szinten eltérő* memóriatitkosítást, azaz *minden futtatott virtuális gép (guest)* eltérő titkosítási kulccsal kezelheti a *saját* memóriáját. Bár ezt elvileg egy külön

processzormag kezeli, ami nem hozzáférhető, aggodalomra ad okot, hogy *ebben is találtak azóta sebezhetőséget*..

Egy *virtualizált gép* esetén eddig is ismert probléma volt, hogy a *gazdagép* (tehát a virtuális gépek futását biztosító fizikai számítógép) tulajdonosa *rálát* minden adatra, amely a virtuális gépben tárolásra vagy feldolgozásra kerül. Ez ugye egy saját fizikai gép esetén másodlagos probléma, hiszen az üzemeltető szükségszerűen azonos. Ezen hardver-sebezhetőségekkel azonban emellett megjelent az a probléma is, hogy *a mai, virtualizációhoz használt hardvereken nemcsak a gazdagép, hanem a többi virtuális gép üzemeltetője is kockázatot rejthet az igénybevevő számára*. Ráadásul egy bármilyen adatszivárgás esetén a gazdagép üzemeltetője *mutogathat* bármelyik másik *bérlőre* (akinek a kiléte üzleti és adatvédelmi titok), azzal, hogy *általa nem elhárítható okból* történt a szivárgás; és konkrét bizonyítékok nélkül ezt nehéz lehet megcáfolni. Felelős pedig a legtöbbször nincsen – az *automatizáció* kizárja, hogy előre lehessen tudni, hogy ki fogja a következő *virtuálisgép-szeletet* a fenntartott állományból igénybe venni (mint ahogy azt is, hogy 30 perc múlva még mindig ugyanaz-e az igénybevevő).

## 3.2. Szoftver-, webalkalmazás-fejlesztési trendek, módszertanok, technológiák

### 3.2.1. A vízesés-modell

Itthon és külföldön is elsőként a vízesés fejlesztési módszertan terjedt el. Kialakulásának vélelmezhető oka, hogy hagyományosan, az objektum-orientált megközelítés elterjedése előtt a *procedurális programozási* módszer volt elterjedt, amelyben nem voltak *osztályok*, hiányzott az *enkapszuláció*, mint a tagolást rendkívüli mértékben megkönnyítő minta, ezért lényegesen kevesebb érdemi lehetőség volt a fejlesztési folyamat *párhuzamosítható*, *független* részfolyamatokra bontására.
A vízesés módszertanban *egy* tervezési és *egy* fejlesztési szakasz van, végül egy *ellenőrzés (validálás, verification)*, ahol az esetleges *szembetűnő* problémák javítása megtörténhet; minden ezt követő módosítási és egyéb igény az utolsó, *karbantartás* fázisba tartozik.
Royce szerint ez a modell újdonságot jelentett abban, hogy *előre nem látható nehézségek esetén tudta a veszteséget, kvázi a fölösleges, elpocsékolt fejlesztési időt minimalizálni* (ROYCE, W. W., 1970.). A szerző az *ügyfél bevonását*, illetve *kritikai szoftver-vizsgálatot (Critical Software Review)* javasolt a vázolt problémák további visszaszorítása érdekében. Royce vezette be egyébként az *ellenőrzés* fázisát; a gyakorlati tapasztalatok alapján ez a fázis néha elmarad..

Amikor később terjedni kezdett az objektum-orientált fejlesztési módszertan, és lehetővé vált az *interface*-ek segítségével a *cserélhető (pluggable) komponensek* használata, ezzel egyidejűleg megnyílt az a lehetőség is, hogy az egyes komponensek egy előzetes interface specifikációt követően *párhuzamosan*, eltérő környezetben készüljenek. Ezzel lehetővé vált a modell iterációs alkalmazása részfolyamatokra, komponensekre.

A vízesés-modell a mai napig számos helyen használatban van, új projekteknél is, és visszatérőknél is, mivel egyes vélemények szerint az *agilis* módszerek *sem oldanak meg minden problémát* (LINTHICUM, D., 2019).

### 3.2.2. Spirál módszertan

Boehm, aki Royce-al részben párhuzamosan dolgozott a TRW Defense Systems-nél, az 1980-as években ugyanazokat a problémákat kezdte vizsgálni: *a veszélyes, körülményes, és/vagy állandóan változó követelmények problematikáját*. 1988-ban kiadott tanulmányában visszacsatolásokat alakított ki a *vízesés* modell egyes lépcsői között, majd levonta azt a következtetést, hogy a folyamat *egyes* elemei a kialakult lépcsőben *helytelen sorrendben követik egymást* (BOEHM, B. W., 1988). Ebből kiindulva kidolgozta a saját sorrendjét, amelyet egy Descartes-szerű koordináta-rendszeren helyezett el. Ezen a középpontból kiindulva spirális módon, az adott tengelyekre vissza-visszatérve, a középponttól való távolsággal a költségnövekedést szimbolizálva jelent meg a folyamat.

### 3.2.2. Agilis módszertan

A projektkockázatok alapján belátható, hogy minden olyan, a *vízesés* módszertan szerinti fejlesztési projekt, amely nem kellő körültekintéssel és időtartalékkal megtervezett, nem veszi figyelembe a csapatkockázat fejlesztői és ügyfél-oldali szempontjait, különös tekintettel a *bérkényszer,* vagy a *fluktuációt* követő esetleges betanulás időigényének kérdésére, vagy éppen nem áll *előzetesen* rendelkezésre a tervek alapján kalkulált *költségkeret,* a feltárt kockázatok okozta esetleges költségnövekedés igényelte tartalékokkal együtt, jelentős kockázatot hordoz.

Tekintve, hogy az én tapasztalatom az, és a PwC CEO megkérdezéséből is az tűnik ki, hogy földrajzi elhelyezkedésétől és méretétől, tőkeellátottságától *függetlenül* ma a legtöbb vállalat vezetője ezeket *nem képes és a mai környezetben nem is lehet képes b*iztosítani (GLOGER, M. ET AL., 2019), célszerűnek tűnik lépéseket tenni annak érdekében, hogy a problémát a keletkezése során vagy rögtön utána kezeljük.

Ezek egy részére a spirál módszer megoldást nyújthat. Az agilis módszertan azonban ennél is messzebbre megy. Bár a Kiáltvány alapján az elsősorban az igények és a fejlesztési feladatok közötti, *félreértéseken alapuló* eltérések minimalizálását és ezáltal az ügyfél-elégedettség növelését tűzte ki célul (BECK, K. ET AL., 2001), a *folyamat sajátosságaiból* következik, hogy a feltárt kockázatok közül *számosra* megoldást jelent, míg *a többi* esetében segít, hogy például a Lidl SAP-fiaskójával szemben ne évek múlva, hanem akár 1-2 héten belül fény derüljön. Külső szemlélő számára úgy tűnhet, hogy *az agilis módszertan tulajdonképpen a spirál egy bővített változata;* ezt az érintettek úgy cáfolják, hogy a megközelítésbeli különbséget (*folyamatközpontúság helyett emberközpontúság)* hangsúlyozzák.

Sajnos személyes tapasztalatom az volt, hogy itthon rengeteg, az információ-technológiában tevékenykedő, fejlesztő mikro- és kisvállalkozás egyfajta *sajátos torzszülöttet* alakított ki saját folyamatai között: *összeülnek, beszélgetnek hetente*, de az

ügyfelet *nem vonják be* a folyamatokba, a tapasztaltak szerint jellemzően olyan okokból, hogy *az ügyfélnek mindig igaza van, túl elfoglalt, nem ér rá "ilyesmire"*, és hasonlók.

Bár vannak olyan szektorok, ahol ennek van létjogosultsága (például a piacra termelő cégek esetében a terméket a *validálás* előtt kell elkészíteni – ilyen esetben a *belső vevők* biztosítása az indokolt), a többi esettel kapcsolatosan itt és most fontos leszögeznem két dolgot.
Egyrészről *a vevőnek mindig igaza van* gondolatmenet a *hagyományos* kereskedelemben teljesen jól megállta a helyét, hiszen minden eladott termék, szolgáltatás *értéknövelt* volt; a hozzáadott érték a növekedés biztosításán felül számos, a legtöbbször valódi szolgáltatást is tartalmazott. Ugyanez a megközelítés kevésbé létjogosult olyan környezetben, ahol több árajánlat bekérését követően szükségszerűen a legkedvezőbb költségvonzatú érvényesül.
Másfelől ugyanakkor az, hogy a *megrendelőnek nincs ideje a saját projektjéről egyeztetni*, *nonszensz*. Ez előfordulhat olyan esetben, ha az érintett például egy nonfiguratív festményt vagy plasztikát szeretne a nappalijába, hiszen ott kevés kivételtől eltekintve nem értelmezhető a *célra való alkalmasság* kérdése; egy fejlesztési projektben azonban az összes *termék*- és *projektkockázatot* magában hordozza.

Az ilyen módon, a megrendelő bevonása nélkül történő *"agilizálás"* nem alkalmas sem az agilis módszertan eredeti fő céljának, az ügyfél-elégedettségnek a javítására, sem pedig a *termékkockázatok* csökkentésére. Legfeljebb a munkaidőben videót néző munkatársak kiszűrésére lehet alkalmas, ez azonban célszerűtlen is lehet, ugyanis a hasonló *komolyságú* foglalkoztatók vélhetően lényegesen nehezebben fognak tudni bármilyen szempontból elkötelezett munkavállalókat alkalmazni.
Az egyetlen, amire ez a *kifacsart* megközelítés alkalmas lehet, az az, hogy a fejlesztést végző szervezet *illúziókat létesítsen a saját módszertanának nem létező garanciáiról*.
Remélhetőleg nem szükséges bemutatnom, hogy ez miért lehet *rosszabb, mint a korábbi, esetlegesen kaotikus állapot*.

Egy jól kivitelezett agilis megvalósítás előnyei tehát egyértelműen kimutathatóak a vevő szempontjából, és a fejlesztőknél is hasznos, hogy az egy-két hetente *kötelező* egyeztetéseken szükségszerűen kiderül, hogy/ha valamilyen funkció, folyamat nem a másik fél által kívánt irányba tart.

Mindezek mellett van azonban egy, az eredeti elvekbe nem beleértett, azonban azokból egyértelműen következő *üzleti* előnye is: *a szállítói teljesítéshez hasonlóan a számlázás is szükségszerűen ciklikus*, vagyis az ügyfél legfeljebb a legutóbbi ciklus (jellemzően egy-két hét) termésére hozhatja fel, hogy *ő nem erre gondolt*, amely így csökkenti az esetleges kiegyenlítetlen számlákból adódó vitatott követelések létjogosultságát. Természetesen megállapodáson alapuló módon ettől el lehet térni, de a hazai helyzetet ismerve ez nem okvetlen bölcs dolog.

Látható, hogy *kellő óvatosság* alkalmazásával a korábban feltárt kockázatok milyen magas része ellen biztosít ezen megközelítés vagy egyenesen *védelmet*, vagy legalább

az előzetes *rálátás*, *monitorozás lehetőségét*, amely teret biztosít az *informált beavatkozásnak*, mielőtt az esetlegesen bekövetkező kár elérhetné a visszafordíthatatlan mértéket.

### 3.2.3. SOLID irányelvek

A szoftverfejlesztésben, minden bizonnyal a Java programnyelv terjedésével és a személyi számítógépek, valamint az Internet terjedésével párhuzamban, az agilis módszertan kibontakozását közvetlenül megelőzően több módszertani változás, ha úgy tetszik, *mozgalom* is szárnyat bontott. Az agilis mozgalom egyik résztvevője, Robert C. Martin 1997-2000 között több könyvet is írt a *szoftverarchitektúra* tárgykörében. A sorozatban legutolsó, saját honlapján közzétett könyvében összefoglalta a korábban külön-külön könyvekben is tárgyalt, úgynevezett SOLID irányelveket (MARTIN, R. C., 2000). Ezek jellemzően a szoftverek architekturális tervezése során, a *modularitás*, részegységekre bontás problémáit járják körül.

Ezek közül kiemelnék kettőt, amely véleményem szerint a legkritikusabb, illetve amelyek figyelmen kívül hagyása a legtöbb problémát okozza. A többi három közül kettő magasabb szintű architekturális kérdés; megvalósításukat a legtöbb webes keretrendszer alapértelmezetten támogatja; vagy például a Liskov-féle helyettesíthetőségi elvhez szükséges *visszatérési kovariancia* és *paraméter-kontravariancia* deklaráció szerinti *kikényszerítését* nem is mindegyik programnyelv vagy környezet biztosítja.

### 3.2.4. S: szétválasztott felelősség elve (Single Responsibility Principle)

Ezek közül véleményem szerint a *legfontosabb* a legelső, a SOLID-ből az "S": *Single Responsibility Principle*, ami tartalmi hűséggel magyarra körülbelül a *szétválasztott felelősség elve* lehetne. Lényege, hogy *bármely modul, komponens, folyamat, vagy akár megközelítés egyetlen, jól körülhatárolható (karakterizálható) dologért legyen felelős – és csak azért*. Ez az egyetlen az öt közül, amiről a szerző nem írt külön könyvet – ez pedig arra vezethető vissza, hogy ez az elv megegyezik a némileg tágabb *érdekeltségek elválasztása* (*Separation of Concerns, SoC*) elv adott területre leszűkített változatával. Ez utóbbi fogalmat először még a strukturált programozás egyik atyjának tekinthető *Edsger W. Dijkstra* alkotta meg a hetvenes évek derekán (DIJKSTRA, E. W., 1974).

Amint azt tapasztaltam, tőlünk nyugatra már egy évtizede, de itthon is az utóbbi négy-öt évben jellemző, hogy ha nem pályakezdő vagy junior, hanem magasabb szintű fejlesztői pozíciót szeretnének betölteni, a jelölttel szemben elvárásként jelenik meg a SOLID irányelvek ismerete. Felettébb megdöbbentő módon azonban ez egyfajta *többes személyiség*-jelleggel jelenik meg – sokan vannak abban a hitben, hogy jól értik e két, egymásra épülő elvet, miközben *alapvető dolgok fölött* hajlamosak elsiklani.

*Nézzünk két, nem mellőzhető példát.*

#### *A forráskódon belüli érdekeltségek kérdése*

Az első és leggyakoribb a *forráskód* kialakítása, elrendezése terén jelentkezik.

Természetesen sok helyen van súrlódás abból, hogy a fejlesztők eltérően szokták meg a kód formázását. *Lényegesen* nagyobb probléma, hogy sokan, köztük *programnyelv- és keretrendszer-alkotók és fejlesztők,* összekeverik a forráskód elemeinek, elrendezésének a felelősségi kérdéseit.
Szakmai körökben viszonylagos konszenzus van abban, hogy *a jó programozó áttekinthető forráskódot ír.* Mint Martin Fowler, a tiszta kód (Clean Code) mozgalom egyik atyja írta a *refaktorálásról* szóló könyvében: *"Bármely bolond tud olyan kódot írni, amelyet a gép megért. A jó programozó olyan kódot ír, amelyet az emberek [is] megértenek."* (FOWLER, M. et al., 1999, p. 22.). Habár én a bolondok és a gép általi érthetőségét nem tudom bizonyítani, az állítás második részével nem tudok vitatkozni.

Ugyanakkor álláspontom szerint a nagy olvashatóság iránti hajszában valahol kiveszett a korábbi bölcsesség. Ugyanis attól, *hogy a forráskód az emberek számára érthető,* attól még – jelen elv értelmében – az ehhez szükséges műveletek *elválasztandóak* a *gép általi* érthetőséghez szükségesektől. Például a *Python* programnyelv (amely egyebek mellett neurálisháló- és egyéb AI-könyvtárak általi támogatottsága miatt egyre népszerűbb), *szóközöket használ a szintaxis-fa elemeinek, tehát az úgynevezett scope-oknak (például egy elágazás, vagy egy metódus törzsének) az elválasztására*. Ez azzal jár, hogy valamely blokk sorait *kötelező* az előző, ugyanoda tartozó sorral azonos mélységben tagolni (*indent*); a "rossz" tagolás *feldolgozási hibához* vezet.

Ehhez hasonló a Google új, *Go* nevű nyelve: a sok tekintetben a C-n alapuló szintaxisú nyelv *kötelezővé tette*, hogy a nyitó *brace* – { – karakter *az azonos sor végén* legyen.
Ez egyértelműen a Single Responsibility elv megsértése, hiszen *a kód whitespace tagolása* egyértelműen *az embereknek szól*, míg a fordítói tokenizáció *a gépnek;* jelen elv értelmében *ezek teljesen elkülönülő területek;* vagyis a fordítónak a *soronként egy tokent tartalmazó forráskódot ugyanúgy fel kell tudnia dolgozni, mint az esetlegesen egyetlen sorba írtat;* és *az értelmezőnek nem feladatköre az olvashatóságot, pláne egy adott kódolási stílust számonkérni a programozón.*

Az iparági törekvések ráadásul egyértelműen abba az irányba mutatnak, hogy a forráskódok *keresztfordíthatóak* (*transpilation)* legyenek, vagyis egy (gyakran magasabb szintűnek tartott) nyelv forráskódja átfordítható legyen egy másik (esetleg gépközelibb) nyelvre. Nem igényel feltételezést az sem, hogy milyen *hatékonysággal* járhat, ha az ilyen *köztes* fordítások *akár több megabyte* szóközt, tabulátort, vagy egyéb *tetszőleges karaktert kell* tartalmazzanak, pusztán azért, mert valamely fordító túllépett az *érintettségi* keretein, és ilyen elvárásokat támaszt.

A Google-nél egyébként messze nem ez az egyetlen, szakmai szempontból minimum kifogásolható jelenség mostanában. Lassan 6 éve, hogy megjelentek a piacon egy új programozási nyelv, a *Dart* 1.0-s változatával (a beharangozóra már két évvel korábban, 2011-ben sor került). Akkor egyes orgánumok ezt egyenesen a *JavaScript utódjának* hirdették (BOLTON, D., 2019). Voltak hangok, amelyek már ekkor *óvatosságra intettek*, végül a Google 2015-ben megszüntette az ezt támogató *Dartium* böngésző fejlesztését. A nyelv azóta tavaly új erőre kapott, egyebek mellett a *Flutter* mobilalkalmazás-

keretrendszernek köszönhetően, és idén megjelent belőle a 2.0-s változat, sőt a napokban jött a hír, hogy *natív, gépi kódot előállító fordítót* is kapott a nyelv – az áttörés azonban várat magára. Ez álláspontom szerint nem független az egyik fejlesztő attitűdjétől, aki elkészítette ehhez a nyelvhez a *hivatalos forráskód-formázót*, amely a *dartfmt* névre hallgat. Ez az eszköz a *hivatalos ökoszisztéma része*, szándékosan *nem konfigurálható (az alkotója szerint "meglepő lehet", de az eszköznek nem az a célja, hogy a forráskód a fejlesztő elképzelése szerint nézzen ki),* és bár a saját dokumentációja *(DART STYLE FAQ)* szerint *nem való mindenkinek; ha nem tetszik, nem kötelező használni* , a "Hatékony Dart" útmutató már *kötelezőként* jelöli meg a használatát, és mindenképp meg akar győzni arról, hogy *mekkora szerencse,* hogy a formázással *nem kell törődnöm*.. Ehelyett valójában a Single Responsibility alapján *nekem* nem kell törődnöm azzal, hogy a nyelv alkotói hogyan optimizálják a futó gépi kódot, *a nyelv alkotóinak* pedig nem kell törődniük azzal, hogy én hogyan formázom a kódom.

Bár kifejezetten az a véleményem, hogy a Dart egy *jó nyelv* – tetszetős szintaxissal, generikus típusok támogatásával, egzakt típusossággal, más nyelvek számos problémájának kiküszöbölésével -, az ilyesfajta attitűdöt öncélúnak és kizárónak tartom. Ezzel olyannyira nem vagyok egyébként egyedül, hogy a szintén "házon belül" fejlesztett *Flutter* keretrendszer – amelynek a Dart kvázi *reneszánszát* köszönheti – fejlesztőinek is meggyűlt a baja ezzel az eszközzel, és végül semmilyen kompromisszumot nem sikerült kötniük a kódformázó fejlesztőjével.

Sajnos ez a jelenség rávilágít egyes, a fejlődés szempontjából meghatározó szerepet játszó piaci szereplőnél zajló folyamatok egy nem elhanyagolható részének problémáira is, amely vélelmem szerint nem mentes a csapatkockázat okozta kontraszelekciótól.
Ez feltehetőleg arra vezethető vissza, hogy az *új generáció kellő szabadságot* kapott a különféle projekteken való munkálkodáshoz, eközben azonban mintha nem fordítanának kellő energiát a korábbi, hosszú évek alatt összegyűlt ismeretanyag és koncepciók áttekintésére és figyelembe vételére.

Természetesen vannak ennél némileg földhözragadtabb példák is, a legjellemzőbb a különböző programnyelvekben fellelhető annotációk kérdése. Határeset, hogy a Java fordító esetében az annotációk, mint tájékoztató információk elvárása (pl. egy "Override" esetén) ezen elv megsértése-e, vagy nem (szerintem igen, tekintve, hogy ezen információ nem szükséges a kód értelmezéséhez; de pl. egy figyelmeztetés indokolt lehetne).

De például C#-ban annotációk helyett úgynevezett attribútumok vannak, ezeket pedig nagyon sokan szinte visszaélésszerűen használják úgynevezett deklaratív programozási célkitűzések elérésére. Amennyire vonzó, hogy csak néhány függvénykönyvtárat kell betölteni, majd ellátni néhány *attribútummal* a metódusokat, és máris, *mágiaszerűen*, működik a program, annyira megtévesztő – hiszen egy ilyen termék elkészültét követően a legtöbbször nemcsak a termék, hanem az adott komponens működésével kapcsolatos *problémakör-specifikus* tudás birtokában *sincs* a fejlesztő. Elgondolkodtató,

vajon mennyire előnyös így egy esetleges ügyfél-megfelelőségi probléma esetén váratlan külső kérdésekre válaszolni..

A PHP nyelvben a probléma még nagyobb: például a Symfony nevezetű (szakmai körökben a második legelterjedtebb, emellett a legrégebbi és legátgondoltabbnak tartott) webes keretrendszerben számos funkció, például a vele szállított *Doctrine 2 objektum-reláció-leképezés könyvtár (object-relational mapping, ORM)* meződefiníciói, vagy éppen a *vezérlők eseményeinek az összerendelése (routing)* is *történhet annotációkkal*.
Ezzel a legnagyobb probléma, hogy a PHP alapból *nem támogat* annotációkat, azok *virtuálisan* léteznek, gyakorlatilag teljesen kódközi *megjegyzésekként*.

Egyszóval ma *bármikor* előfordulhat, sőt *már-már megszokott*, *szinte bármely programozási nyelvben*, hogy egy adott forrásállomány *végrehajtható kód része* különböző külső felek és emberek gyakorta öncélú elvárásainak kell, hogy megfeleljen elrendezés és tagolás szerint – mialatt a *fordító* – vagy akár éppen a *futtatott kód* saját magában – keresi a *neki szóló konfigurációs részleteket*, hol máshol, mint *a megjegyzések között* – amelyek hagyományosan az emberi szereplőknek vannak létrehozva.

*Ha ez nem mond ellent a szétválasztott felelősség elvének, akkor nem tudom, hogy mi lehet az, ami ebbe ütközne.*

Emellett az ötlet, hogy az adott *vezérlők* maguk *mondják meg*, hogy milyen bejövő webcímekre kívánnak válaszolni, mindenféle külső beállítás nélkül, az felveti ezen elv mellett a *függőség-megfordítás* csorbulását is.

### *A vélelmezett kézenfekvőség problémája*

Bár az előzőekben körüljárt probléma sajnos lassan a teljes webes ökoszisztémát átjárja, ezért muszáj volt ekkora terjedelmet szentelni neki, a fejlesztők többségének azért jó eséllyel nem ez a leggyakoribb érintettsége a szétválasztott felelősség kapcsán.

A leggyakoribb probléma, hogy valamely tervezési minta projektszintű használata során az azzal ismerkedő, vagy azonosulni nem kívánó fejlesztő az azzal járó *komplexitást* megkerülve, annak keretein *átnyúlva* valósít meg valamely működést. MVC esetén tipikusan ilyen a *vezérlőbe* vagy a *nézetbe* ágyazott *adatbázis-lekérdezés.*

Tervezési minta nélküli fejlesztés esetén még kuszább megvalósítások tudnak napvilágot látni. Személyes kedvencem az volt, ahol a fejlesztő az adatbázisban előforduló, *nem felhasználó-specifikus* adatok egy részét *a felhasználó-specifikus munkamenetben (session), a memóriában tárolta.* Amellett, hogy a munkamenet *nem ezt a célt hivatott szolgálni*, nem nehéz belátni, hogy 256 kilobyte adat ilyen módon történő "*gyorsítótárazása*" 10000 látogató esetén körülbelül 2.4 GB memóriahasználattal járt (ugyanaz a 256 kB lett tízezerszer letárolva, természetesen).
Természetesen nem tízezer *egyidejű* látogatóról beszélünk, hanem a munkamenet 2 órás

élettartama alatti látogatóról – ideértve a különböző
Gyorsnak gyors volt, persze. Olcsó nem volt.

De ugyanide sorolható valamely *relációs adatbázisban* hatalmas karakterlánc-mezők létrehozása különféle *SOAP* vagy *ReST* szolgáltatások *XML*, *JSON*, ... válaszainak komplett tárolására, majd minden egyes lekérdezésnél, listázásnál az ezeken történő karakterlánc-műveletek (*SUBSTR*, ...) végzése is. *Ha ez kézenfekvőnek tűnik, az probléma – de még nagyobb, ha meg is valósul.*

### 3.2.5. O: A nyíltság – zártság elve

A második, zöldmezős fejlesztésnél figyelembe veendő szempont a *nyíltság – zártság szempontja*. Ez azt mondja ki, hogy egy adott szoftver-egység (osztály, metódus, *interface*, stb.) *legyen nyitott a kiterjesztésre, de elzárt a módosítás elől.*

Ennek – a korrektség, a harmadik fél által fejlesztett könyvtár és a saját kód közötti érdekeltség-szétválasztás, illetve a *purizmus* mellett a legnagyobb és egyben legkönnyebben feldolgozható előnye, hogy *az adott könyvtár a támogatási ciklusán belül frissíthető marad*. Érdemes elképzelni, hogy a fejlesztő félig átír egy kész, "dobozos" mikroszolgáltatás-kiszolgálói réteget, aztán jön a hír, hogy súlyos sebezhetőséget találtak benne, és az igen szorgalmas készítő már ki is javította, ráadásul *ingyen* – de ehhez a kód felét le kellett cserélni... *Valóban előre nem látható kockázat?*

Ez teljesen kézenfekvőnek és egyszerűnek tűnhet elsőre, a probléma itt is értelmezési kérdésekben van. Az első és legfontosabb, hogy a *módosítási korlátozás mindig projektek között értendő*. Tehát a szabály szerint alapvetően *semmilyen indokkal nem írunk bele a használt függvénykönyvtárba akkor sem, ha nyílt forráskódú;* a szükséges módosításokat az osztály kiterjesztésével és az érintett metódusok felüldefiniálásával oldjuk meg.

Sajnos ez nem minden esetben kivitelezhető – a leggyakoribb akadály az, hogy *maga az érintett könyvtár vagy osztály* úgy lett elkészítve, hogy nem támogatja az ilyen használatot. Számos esetben fordul elő, hogy *lezárt (sealed) / végső (final) osztállyal* találkozunk, amelyet nyilván nem lehet kiterjeszteni; szintén gyakori, hogy az adott *védett* (*protected*) metódus *privát tulajdonságokat használ*, amelyeket a kiterjesztett példányból nem lehet elérni. Bár ezt a *privát tulajdonságok szorgalmazását* több iskola terjeszti, különféle, általam nem teljesen követhető, a *tesztelésre* hivatkozó érvelés mentén, a gyakorlat azt mutatja, hogy ezek az osztályok *ellehetetlenítik a nyílt-zárt elv szerinti kiterjesztést*, és ezáltal az osztályok kevésbé követhető módosításaihoz, összekuszálásaihoz vezetnek.

### 3.2.6. Tervezési minták

Az agilis módszertan megjelenésénél jóval korábban, a szoftverfejlesztés kezdete óta megvoltak az igények a moduláris, újrahasznosítható szoftverkomponensek meglétére. Ez a kezdeti időkben a függvénykönyvtárak szintjére korlátozódott. Az első objektum-orientált fejlesztésre alkalmas nyelvként az 1967-ben megjelent *Simula*-t tartják nyilván,

a *paradigma-váltást* azonban az 1972-ben hozzáférhetővé vált *Smalltalk* nyelv kezdte el igazán népszerűsíteni. Az objektum-orientált paradigma lehetővé tette, hogy egy adott programon belül is a komponensek *állapottal együtt* (enkapszuláció) többszörözhetőek, átadhatóak legyenek egyéb komponenseknek. Az *áttörés* azonban még váratott magára. Igazi változást a C++ nyelv 1985-es megjelenése hozott, amely megkönnyítette a C nyelvben már járatos fejlesztők egyszerűsített megismerkedését az új megközelítéssel. 1989-ben megjelent a C++ 2.0, rá két évre az alkotó, Bjarne Stroustrup könyvének második kiadása, amely a "The C++ Programming Language" névre hallgatott, és a korai időkben a *szabvány* szerepét is betöltötte.

Ezzel párhuzamosan felerősödtek a törekvések az újrahasznosítható objektumokon alapuló architekturális módszerek továbbfejlesztésére. Gamma és társai (akiket a szakzsargon gyakorta a *Gang of Four*, azaz a "négyek bandája" néven emleget) 1994-ben megjelent könyve *alapműnek* tekinthető; a problémakör körüljárásán és gyakorlati példákon át gyűjtött össze, illetve alakított ki különböző, *egyszerűnek és elegánsnak* nevezett módszereket ezen problémák megoldására, illusztrálva azokat *Smalltalk* és *C++* példakódokkal (GAMMA, E. ET AL., 1994). Az így felsorolt módszereket *tervezési mintáknak (design pattern)* keresztelték, és három fő csoportra osztották ezekat: *létrehozási – példányosítási (creational), szerkezeti(structural)* és *viselkedési (behavioral)* minták.

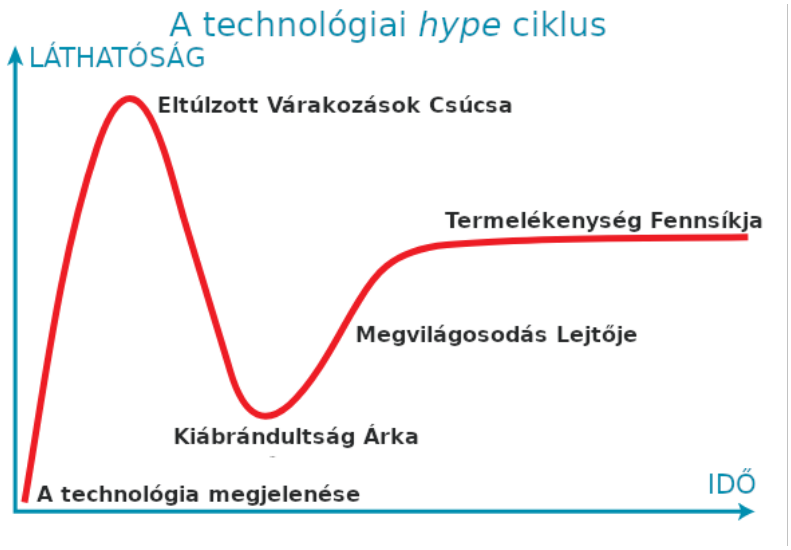
Ezekkel párhuzamosan több más *mintacsoport* is megjelent. Az egyik legfontosabb az *architekturális minta*, amely a projekt teljes szerkezetét *alapjaiban* meghatározza. A ma egyik legnépszerűbb ilyen minta a *modell-nézet-vezérlő (MVC)* minta, amely lényege, hogy az alkalmazás három részre van bontva: a *modell* tartalmazza az üzleti logikát, tehát az adatok közötti relációt, illetve az adatkapcsolati réteget; a *nézet* a megjelenítéshez kapcsolódó dolgokat foglalja magába; míg a *vezérlő* felelős azért, hogy a *nézetből* érkező utasításokat továbbítsa a *modell* felé, illetve a *nézet* által reprezentált adatokat a *modellből* lekérdezze és továbbítsa a *nézet* részére. Ezt a mintát Trygve Reenskaug alakította ki, aki a *nagy és komplex adathalmazok felhasználók általi kezelési* problémája egy *általános megoldásának* szánta (REENSKAUG, T., 1979). Ezen minta egy változata az 1980-as évek elején megvalósításra került a Smalltalk-80 dialektus osztálykönyvtárában, de *akadémiai koncepció* szintjén először Krasner és Pope 1988-as tanulmányában jelent meg (KRASNER, G. E., POPE, S. T., 1980)
Ennek egy módosított változata az *MVVM (modell-nézet-"nézetmodell").* Amint azt Reenskaug is említette, "*az MVC problémának több arca van*", mint amivel ő számolt 1979-ben. Az egyik ilyen, hogy *az adatok nem szükségszerűen olyan formában vannak tárolva, hogy az közvetlenül a megjelenítésre alkalmas* legyen. Ilyenkor érdemes beiktatni egy közbeeső réteget abból a célból, hogy az adatokat *a megjelenítéshez, könnyű eléréshez* alkalmas módon *átalakítsa (transzformáció).*
Itt fontos megjegyezni, hogy létezik az úgynevezett *adattár (repository)* minta, amely az MVC egy másik kiegészítéseként, párhuzamosan is alkalmazható; nevével kvázi ellentétben ennek fő célja, hogy a konkrét adatbázis-kapcsolati réteget *leválassza* a *modell* üzleti logikájáról. Ilyenkor minden *perzisztencia-specifikus* elem (kiváltképpen

az esetleges SQL lekérdezések) az *adattár* rétegbe kerül, és a *modell* ennek definiált *interface-ével* kommunikál.
Bár én magam kifejezetten támogatom a *repository* minta használatát, *igen fontos*, hogy a fejlesztő bármennyire is kézenfekvőnek találja esetleg, hogy *a megjelenítés számára fontos* adatváltoztatásokat (*különösen: aggregált attribútumok szintézise több, valódi attribútumból;* vagy például *számok szöveggé alakítása* az érthetőség szempontjából) az *adattár*-rétegbe tegyen, ezt *mindenképp* szükséges elkerülni, hiszen ez sértené a *szétválasztott felelősség* elvét. Szükség, illetve *igény* esetén tehát *különálló viewmodel-, és adattárház-réteg létesítése* a megfelelő megoldás.

### 3.2.7. A csapatkockázat és a guru-jelenség technológiai problémái

Amint az az előző alfejezet adataiból kitűnik, magam és az érintettek is *dominánsnak* tekintik a *csapatkockázatot* valamely folyamat lehetséges kimenetelének véleményezése során. Sajnálatos módon a szoftverfejlesztők munkaerő- és közvetve szolgáltatás-piacán, elsősorban a rendkívüli módon megnövekedett kereslet, illetve a *költségkényszer* miatt, emellett pedig számos egyéb, feltárt és esetlegesen nem vizsgált okból egyaránt, kialakult egyfajta *kontraszelekció*.

Ennek *fejlesztői szemmel* legszembetűnőbb vetülete, hogy *gombamódra* jelennek meg újabbnál újabb technológiák, amelyekhez a fejlesztője *nagy reményeket* fűz, azonban amelyek bevezetése jelentős kockázatot hordoz magában.
E kockázat forrása kettős: egyfelől az új megközelítések sajátossága, vagyis *a kézenfekvő (straightforward) jelentésének újradefiniálása* játszik jelentős szerepet, másfelől az egyre rövidülő ciklusidejű *kiadás-iterációk* – kellő humánerőforrás-ellátottság mellett – a *unit tesztek* megírására akár kielégítő mértékben is keretet biztosíthatnak, azonban az olyan lépések, mint a *dokumentáció-készítés*, vagy éppen a *piaci validáció*, igen gyakran jelentős késést és/vagy kiesést szenvednek.




*2. ábra: A Gartner-féle technológiai hype ciklus*
*Saját grafika Gartner nyomán*

Ez a jelenség nagyjából párhuzamosan alakult a szoftveripar Gartner-féle *hype* ciklusával, és Janes és Succi szerint Nyugaton 2012 táján érte el a "*kiábrándultság árka*" fázist, amely a Gartner-modell szerint az *eltúlzott várakozások csúcsát* követő szükségszerű állapot (JANES, A., SUCCI, G., 2012). A *hype* ciklus Gartner szerinti folyamata a *2. ábrán* látható. Ennek okaként a szerzők – véleményem szerint rendkívül találóan – a *guru-jelenséget* nevezik meg.

A *guru* egyik fő ismérve, hogy *a bölcsesség birtokosaként* – például egy tudományos munkával szemben – *nincs rászorulva*, hogy állításait alátámassza *érveléssel*, *statisztikai* adatokkal vagy bármilyen *egzakt* módon. Ennek megfelelően követői is az olyanok közül kerülnek ki, akik erre nyitottak, fogékonyak; vagyis *nincs szükségük* holmi logikus érvelésre. Őket a szaknyelv *evangélistának* hívja (LUCAS-CONWELL, F., 2006).

Az *evangélista* óriási előnye, hogy a lelkesedés hajtja őket, "az álmot értékesítik". Ez a technológia előállítóinak óriási előny, hiszen egy, a tömeg által *karizmatikusnak vélt guru* technológiáját a lelkes tömegek *teljesen önként, ingyen, ellentételezés nélkül* tehetik világhírűvé. Ám éppen ez a legnagyobb veszélye is: az *evangélisták* többsége rálátás, megértés, mélységi ismeret nélkül terjeszti az általa jellemzően, de nem szükségszerűen kipróbált technológiát – tehát *érzelmi alapon*.

A PHP nyelv például sokak szerint annak köszönheti a töretlen népszerűségét, hogy a *WordPress* nevű, ingyenes néhai blogmotor (ma *tartalomkezelő rendszernek* nevezik) PHP nyelven íródott. A WordPress a W3Techs legfrissebb adatai szerint bolygónk *összes* nyilvánosan elérhető weboldalainak a 35.1%-át, az *ismert tartalomkezelőt* használó weboldalak 61.8%-át szolgálja ki.
Érdekesség a Wordpressről, hogy *eredetileg b2/cafelog* néven született, és egy Michel Valdrighi nevezetű francia fejlesztő munkája volt. Valdrighi azonban, úgy tűnik, *nem volt eléggé karizmatikus gurunak* – 2002-ben leállította a fejlesztést, amelyet 2003-ban átvett *Matt Mullenweg* – úgy tűnik, ő jobban értett az *evangélisták* nyelvén. Innentől a többi már történelem. Az azonban belátható, hogy a népszerűség és a racionalitás nem szükségszerűen rokon területek.

Ez a rövid kitérő mutatja, hogy *valahol* érthető, hogy – amint Lucas-Conwell, illetve Janes és Succi is kiemelik – a technológiai szereplők olykor *hanyatt-homlok* törik magukat, hogy *nehogy kimaradjanak* az aktuális hype-ból. Ezt a szakirodalom FOMO-nak (*fear of missing out*; a kimaradástól való félelem) nevezi.

A Gartner-féle koncepció egyik lényege: a *hype*-jelenségre egyértelműen jellemző a ciklusosság, az ismétlődés. Amint a neves filozófus, Hegel rámutatott, az emberi szellem és az egyed fejlődése (és véleményem szerint *induktíve* mindazok a folyamatok, amelyeket a szellem és az egyed, vagy azok csoportja alkot) is szükségszerűen *körkörös* – ez a körkörösség azonban *nem mentes az érzékelés, kiértékelés* során begyűjtött *tapasztalatok* feldolgozásától (HEGEL, 1807, pp. 117-119).
A technológiai (és üzleti) *hype* ciklusok egy jelentős (és növekvő) része, kívülről nézve legalábbis, mintha mellőzné ezen *visszacsatolási mechanizmusok* többségét.

Amennyiben ez tényleg így van, az ilyen, a visszacsatoláson keresztül leszűrt *következtetéseket, tapasztalatokat nem beépítő körforgás* megalapozottan tekinthető *hiábavalónak* és *öncélúnak* (az eltékozolt erőforrásokat tekintve pedig akár egyenesen *pazarlásnak* is).

### 3.2.8. A JavaScript, mint technológiai kockázat

Minden hálózati alkalmazás, így a *monolitikus* architektúrák is, legalább két fő rétegre osztható: a *kiszolgáló*-oldali (*back end*), valamint az *ügyfél*-oldali (*front end*) rétegre. A web esetén az ügyféloldalon szükségszerűen egy *webböngésző* biztosítja a környezetet a futtatáshoz. Ez azonban számos szempontból problémás.

Miközben a *back end* technológiák – folyamatos fejlődésük mellett – széles választékban, számos nyelven és platformon hozzáférhetőek, és ezek közül számos stabilnak tekintett, ugyanez nem mondható el a *front end* oldalról.

Egyfelől a technológia alapját képező HTML leíró nyelv legfrissebb változatát az úgynevezett Web Hypertext Application Technology Working Group (WHATWG) *kezeli*, amely emiatt nincs kodifikálva; úgynevezett *living standard*. Tovább árnyalja a helyzetet, hogy ezt a munkacsoportot *az Apple, a Google, a Mozilla és a Microsoft* alkotják. Sajnos többször előfordult az utóbbi időkben, hogy a *munkacsoport* a Google vagy valamelyik általal felvásárolt szervezet által (pl. az Angular keretrendszerben) bevezetett, abszolút nem szabványos megközelítéseket *úgyis így fogják használni, írjuk át inkább a szabványt* alapon közelítette meg (WHATWG, 2017).

Ennél nagyobb probléma, hogy a *szerver-oldalon* alkalmazható technológiák *széles tárháza* helyett a böngészőkben egyetlen kvázi-szabványos nyelv érhető el, ez pedig a *JavaScript*. Sajnos tapasztalatom, és a Stack Overflow jelen évi, 88 883 fejlesztőn alapuló adatai alapján is a programozásba *újonnan belépők* számára *a JavaScript a legvonzóbb nyelv* (Stack Overflow Insights 2019). Ennek korrelációját azzal a *ténnyel*, hogy *ez ma a második legkeresettebb* technológia (a Python után), nem vizsgáltam, az azonban észlelhető, hogy *a JavaScript minden eddiginél tágabb teret enged a* magukat a *guru-jelenség* keretében megvalósítani kívánók részére.

Stemmler, aki JavaScript fejlesztő és a SOLID elvekről írt könyvet, rámutat, hogy a JavaScript ma *nem az, ami lenni akart* (STEMMLER, K., 2019). És valóban: a nyelvet eredetileg *Brendan Eich* arra tervezte, hogy – távolról – a Java mintájára alapuló szintaxissal *egyszerűbb* böngészős műveleteket (mint például a feliratok a színek vagy a képek módosítása egy weboldalon belül) elvégezzen. Azóta mind a nyelv, mind az ökoszisztéma egy alapvetően strukturálatlan, kvázi-organikus fejlődésen ment keresztül, aminek eredményeképpen Stemmler szerint a nyelvek *Frankensteinje* jött létre. Bár szinte nap, mint nap jelennek meg újabb funkciók, mivel *a webböngészők tetszés szerinti részhalmaza* a nyelv *szinte tetszés szerinti részhalmazát* támogatja, az újakkal pedig nem kompatibilis, a *közös nevezőnek* a nyelv *ötödik, ES5 (ES=ECMAScript, az ECMA intézet nevéből, amely gondozza)* kódnevű változata tekinthető. Mivel azonban azóta már van *ES6=ES2015* és az azóta eltelt minden évben kijött egy kiadás, ez a

közös nevező jelenleg *nyolc-tíz éves* és *öt-hat változattal régebbi*, mint a legfrissebb, *az ES2019*.

Tekintve, hogy a fejlesztők *többsége legacy technológiának* tekinti a nyelv *régebbi* változatait, a legújabbat viszont például nem támogatja jelenleg semmi, ezért megoldást kellett találni ennek áthidalására. Ez lett az úgynevezett *keresztfordítás (transpilation)*. Lényege, hogy egy, ma már jellemzően JS-ben írt fordító *lefordítja a JavaScript-et JavaScript-re* – de például az ES2019-es szintaxist ES2017-re.
Ez a megközelítés *teljesítmény-szempontból* pont annyira optimális, amennyire első olvasásra tűnik, ez azonban nem meríti ki a problémákat. Ma jellemzően nem *fordítók* vannak erre a célra, hanem *konverziós könyvtárak*, amelyek valójában inkább *programok*, és amelyek *bővítményként (plugin)* tartalmazzák ezeket a fordítókat. Ennek valós előnye, hogy más, például stíluslap-konverziós bővítmények is használhatóak egyidejűleg, illetve más nyelvek JavaScript-re fordítása is lehetővé válik, akár kombináltan is. Ilyen nyelv például a Microsoft által szorgalmazott *TypeScript*, amely a JavaScript típusos változataként fogható fel, de ilyen a Google már említett Dart-ja is.
A legismertebb ilyen konverziós program a *WebPack* névre hallgat, amely a *Babel* fordítót használja JS.hez, és az egész a *Node.js* nevezetű, a Google *V8 motorját* használó *szerver-oldali JS futtatókörnyezeten* fut, és annak *NPM* nevű csomagkezelőjével (vagy a *Facebook Yarn* nevű alternatívájával) telepíthető. Ezzel rögtön nyertünk két újabb problémát. Elsősorban *kénytelenek vagyunk* legalább a fejlesztő-környezetbe Node.js-t telepíteni *pusztán* a JS "*önfordítás*", illetve a stíluslapok konvertálása céljából akkor is, ha egy (vagy több) teljesen más nyelvben fejlesztünk *back end* rétegben. Amióta a Node (és a Babel) megjelent, a korábban a *webre* kialakított eszközök (*Gulp, Grunt, Bower, Browserify,* stb.) *kihaltak;* fejlesztésük megszűnt, tehát itt nincs már sok választásunk; ennél nagyobb probléma, hogy néha úgy tűnik, hogy az *ökoszisztéma*, ideértve a Node.js-t, a csomagkezelőt, és a Babel-t is, *elfelejti*, hogy egyébként nemcsak a Node szerepel a palettán, hanem a böngészők támogatása is *létkérdés*. És ezzel arról a problémáról, hogy egyes nyelvi elemek (pl. az *ES2016 decorator* mintája) *nem valósítható meg* például ES5-ben, csak igen durva közelítéssel szimulálható a JS *metaprogramozási hiányosságai*, például a *tükrözés (reflection)* hiánya miatt, még szót sem ejtettünk..

Az ökoszisztéma azonban ettől függetlenül is problémás: például a Node csomagokat *a hivatalos NPM csomagtárból* lehet telepíteni, amelyben *nem nyilvános* csomagot *díjazás ellenében* lehet tárolni. Nem biztosít ugyanakkor lehetőséget az NPM rendszer arra, hogy egyes csomagokból egy *komplett projekt* valamely *módosított változatot* használjon, csak némi trükközéssel és egy kötelezően GitHub alapú forráskód-tár használatával (*amiből természetesen a nem publikus ismét díjazás ellenében érhető el*).

Mindezt át lehetne ugyan hidalni a *Node* egy környezetiváltozó-paraméterével, ezt azonban a *Babel nem támogatja*, és ami ennél sokkal aggasztóbb, hogy *a fejlesztéséért felelős fejlesztők – miközben egyes bővítéseket arra hivatkozva utasítanak el, hogy a Node nem támogatja – nem mérik fel kellőképpen, hogy mekkora probléma az, hogy ők*

*egyes Node-saját funkcionalitásokat nem építenek be akkor sem, ha valaki azokat elkészíti*.

Ez a gyakorlatban azzal jár, hogy ma, jelenleg, ha valamely JavaScript NPM-csomagon módosítást, kiterjesztést kíván valaki eszközölni, a legtöbb esetben arra kényszerül, hogy – saját NPM (és GitHub) *másolatot* készítsen a kívánt csomagról, és abba a GitHub módszertan szerint elvégezve a fejlesztést, azt utána behelyettesítse a saját projektjébe – *ezzel elveszítve a későbbi javításokhoz, változásokhoz való hozzáférést* -, számos extra feladatot generálva. Emellett elméletben *lehetséges* a helyi, letöltött példányban való módosítás, ez azonban *a szétválasztott felelősség elvének* megsértése, emellett nem biztosítja a *folyamatos integrációnál* megszokott, központi helyről történő hozzáférhetőséget (hiszen *a telepített csomagok a projekt részeként nem verziókövetettek*).

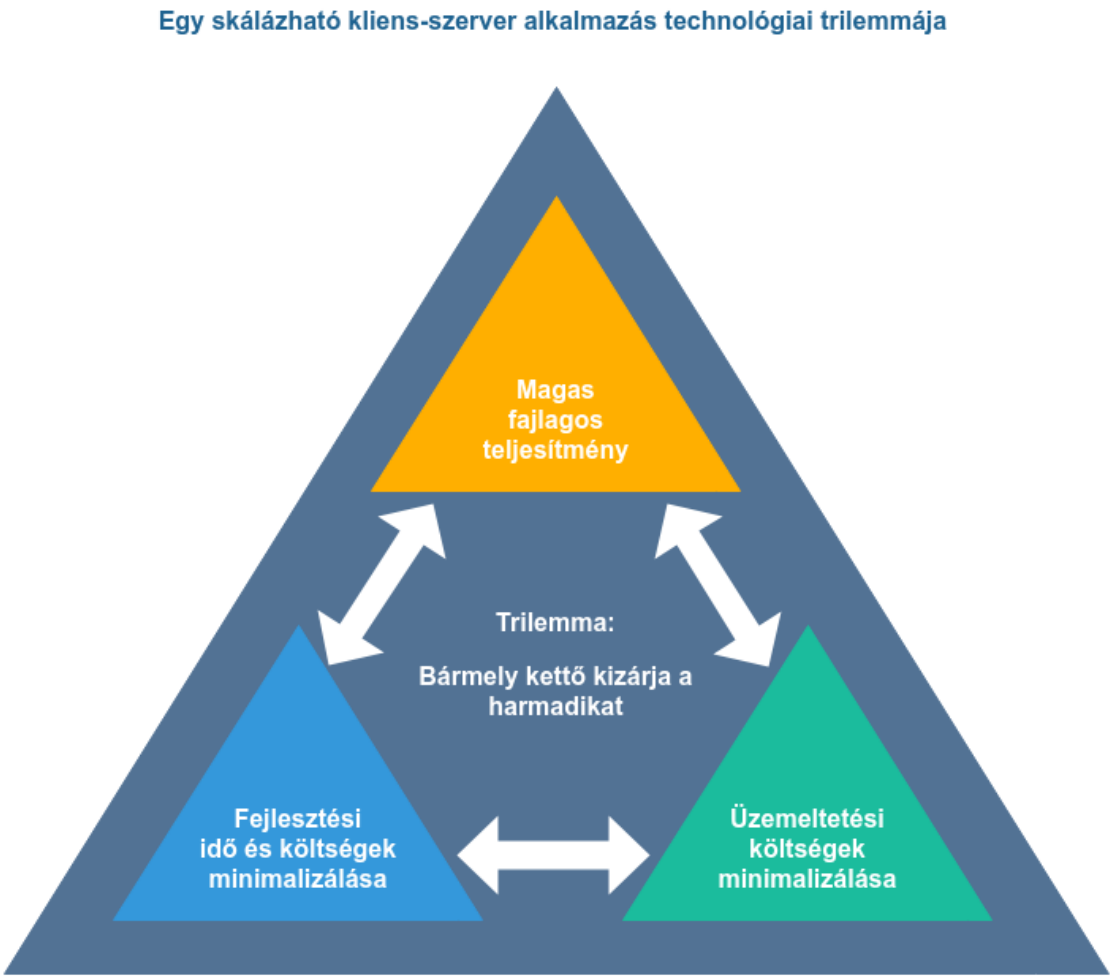
Az általam elképzelhető legdurvább helyzet, ami megtörtént, azonban akkor adódott, amikor az *NPM* üzemeltetője az egyik fejlesztő által korábban beregisztrált nevet egy piaci szereplőnek utólag, igen ellentmondásos körülmények között ítélte oda, mire az adott fejlesztő tiltakozásul az összes, ingyenesen a közösség számára addig elérhetővé tett csomagját törölte (COLLINS, K., 2016). A szövevényes dependencia-hálók és egymásra támaszkodás miatt az egyik, *11 soros* csomag hiányától a *Babel*, a *React (ezáltal pedig a teljes Facebook web UI és a piacvezető csomag többi használójának front end kihelyezési folyamata)* ezáltal *több órára ellehetetlenült*.
Az érettségre és felelősségtudatra jellemző, hogy a *NPM* alapító-CTO-ja, Laurie Voss, a saját felelősségüket teljesen elkendőzve, miután visszaállította az érintett csomagot, azt a kommentárt találta az esethez fűzni, hogy "*egy szerző érdeke állt szemben a közösség szélesebb körű érdekével; az utóbbit választottuk*"..

### 3.2.9. Technológiák kiválasztása

A projektben alkalmazható technológiák közül a *ténylegesen alkalmazottak* kiválasztása során az alábbi szempontok érvényesültek (fontossági sorrendben):

- korábbi tapasztalatok
- a már feltárt kockázatok mérséklése
- törekvés a *modern* technológiák alkalmazására

A korábbi tapasztalatok egy fejlesztési projektnél nem nélkülözhetőek. Amennyiben erre anyagi és időkeret biztosított, magam is a legnagyobb örömmel próbálok ki új módszereket, eszközöket, platformokat – ha az elméleti megismerésük során valamely szempont ettől el nem tántorít -, de *éles* környezetben, *határidők vállalása, becslése* során erről természetesen nem lehet szó. A már (*irodalmi adatok alapján*) feltárt kockázatok mérséklése között kiemelten kezelem a *csapatkockázatot:* szükség esetén lehetőség legyen bevonni fejlesztőt *gazdaságilag indokolható* bérköltség mellett, úgy, hogy ne kelljen objektíven is *idejétmúltnak (legacy) tekinthető* technológiákkal dolgoznia.




*3. ábra: Egy skálázható kliens-szerver alkalmazás technológiai trilemmája*
*Saját grafika*

A *gazdaságilag indokolható bérköltség* kérdéskörénél szükséges még egy pillanatra megállni. A *3. ábrán* látható egy kliens-szerver alkalmazás *technológiai trilemmájának* diagramja. Lényeges, hogy az ábra egy *eleve skálázható* alkalmazásra vonatkozik, vagyis olyanra, amely úgy lett tervezve, hogy *terheléselosztott rendszerként* működhessen, tehát nem küzd szinkronizációs és egyéb problémákkal (amelyek kimerítő tárgyalása meghaladná ezen dolgozat kereteit); ekkor *feltételezhetjük,* hogy az infrastruktúrára fordított (*üzemeltetési*) költség egyenesen arányosnak tekinthető a kiszolgálni kívánt egyidejű kérések számával.
A *trilemma* lényege, hogy bármely felvehető pozíció az ábrán *pontszerű;* bármely két szempont *maximalizálása* szükségszerűen együtt jár a harmadik szempont teljes mellőzésével.A *gazdaságilag indokolható bérköltség* kérdéskörénél szükséges még egy pillanatra megállni. A *3. ábrán* látható egy kliens-szerver alkalmazás *technológiai trilemmájának* diagramja. Lényeges, hogy az ábra egy *eleve skálázható* alkalmazásra vonatkozik, vagyis olyanra, amely úgy lett tervezve, hogy *terheléselosztott rendszerként* működhessen, tehát nem küzd szinkronizációs és egyéb problémákkal (amelyek kimerítő tárgyalása meghaladná ezen dolgozat kereteit); ekkor *feltételezhetjük,* hogy az infrastruktúrára fordított (*üzemeltetési*) költség egyenesen arányosnak tekinthető a

kiszolgálni kívánt egyidejű kérések számával.
A *trilemma* lényege, hogy bármely felvehető pozíció az ábrán *pontszerű;* bármely két szempont *maximalizálása* szükségszerűen együtt jár a harmadik szempont teljes mellőzésével.

Az ábrán észrevehető, hogy *egy tetszés szerinti* előre kitűzött teljesítmény-célértékhez vagy olyan technológiát választunk, amelyben hamar, azaz *költséghatékonyan* lehet fejleszteni, vagy olyat, amelyik *nagy fajlagos teljesítményt* biztosít (természetesen ezek nem *diszkrét* értékek; közöttük tetszőleges átmeneti érték elképzelhető). Ez arra a technológiai körökben széles körben ismert tapasztalatra vezethető vissza, hogy az olyan platformok és nyelvek, amelyek például *szemétgyűjtés (garbage collection, GC)* alapú (vagyis "*félautomata*") memóriakezelést alkalmaznak, jelentősen gyorsítják a fejlesztést, miközben védelmet nyújtanak egyes, memóriakezeléshez kapcsolódó hibalehetőségek ellen (Oracle, 2008); ugyanakkor ehhez cserébe gyakran *többszörös erőforrás-igény* társul (HERTZ, M., BERGER, E. D., 2005). Ilyenkor fontos azt is figyelembe venni, hogy *a hibák,* amelyek ellen az adott technológia védelmet nyújt, *a legtöbbször eshetőlegesek*, míg a teljesítményvesztés *mindig feltétlen és folyamatosan fennáll;* legfeljebb mérsékelni lehet különböző módszerekkel, amelyek gyakran szintén fejlesztés-lassító hatással bírnak.

#### *Kiszolgáló-oldali (back end) technológiák*

Bár van *némi* tapasztalatom *C++* fejlesztésben, és létezik olyan keretrendszer, amely webes alkalmazások fejlesztésére alkalmas, *a kettőt* sosem ötvöztem, és a technológia stabilitásával, adott célra alkalmasságával kapcsolatban sincsen semmilyen tapasztalatom. Emiatt, és a későbbi csapattag-bevonás potenciális szükséglete miatt ezt a vonalat elvetettem.

Főleg az utóbbi szempont miatt a *nem GC-s / nem menedzselt* nyelvek és platformok *nem* jönnek szóba. A legtöbb irodalmi adat és személyes tapasztalatom is a *Java*, *C#*, *PHP* hármasra szűkíti a választékot. Sajnos a C# nagy hátránya, hogy ugyan a Microsoft minden *Linux-pártinak* beállított kommunikációjából inkább az látszik, hogy szeretné az Azure platformjára csábítani a linuxos szolgáltatások üzemeltetőit, a szép reményekkel induló *Mono* platformot továbbra is inkább ellehetetleníteni igyekszik. Hogy ez tudatos-e, vagy sem, nem ismert számomra; mindenesetre amióta megvásárolta a fejlesztőjét, a *Xamarint*, azóta a *MonoDevelop* projekt erős visszaesésnek indult, a *Visual Studio Community-t* nem tervezik Linuxra kiadni (a *VS Code-ot* pedig, amely JavaScriptben íródott, és az *Electron* nevű, szintén a Google V8 motorján alapuló *asztali (desktop)* platformon fut, nem szeretném részletesen véleményezni).

A Java kérdése lényegesen összetettebb. Nyílt forráskódú, elegáns szintaxisú, abszolút korrekt módon megtervezett programnyelv, erős ökoszisztémával és rengeteg iparági tapasztalattal; maga a nyelv tehát minden olyan szempontnak megfelel, amelyet én elvárásként támasztanék vele szemben. Van viszont néhány gond, amelyek ellene szólnak. Az első és legfontosabb, hogy az Oracle kezelésében van. Bár velük kapcsolatban nem merültek fel olyan dolgok, hogy egyik napról a másikra elkezdenének

valami öncélú protokollt vagy stílust rákényszeríteni a fejlesztőkre, a cég híres a *kifogásolható konkurrencia-kezeléséről*. Felvásárolta például a *NetBeans*-t, majd hosszabb időszak után átadta az Apache alapítvány részére. A régi oldalak megszűntek, a bővítmények nem elérhetőek, *jogi* problémák miatt pedig számos funkció nem került még átadásra, vagy a bizonytalanság miatt *elhalt* – így aztán jelenleg a legújabb NetBeans hozzávetőlegesen *fele annyi* technológiát támogat, mint fénykorában, és nem látszik, hogy *merre mutat a jövője.* Magam egyébként NetBeans-használó és -rajongó vagyok, és a kialakult helyzet nem lelkesít. Szintén Oracle probléma, hogy *kihátrált* az *enterprise* környezet (Java Enterprise Edition, EE) mögül (Krill, P., 2017), amely így az Eclipse alapítványához került, és a védjegyjogok miatt újabban a *Jakarta EE* névre hallgat. További problémám a Java alapú webes fejlesztéssel, hogy a legelterjedtebb webes keretrendszer, a *Pivotal* által fejlesztett Spring véleményem szerint *idejétmúlt és tele van indokolatlan komplexitással*, míg "kistestvére", a *Spring Boot* indokolatlan megkötéseket ír elő (mint például a ReST használata).

A harmadik lehetőség a PHP. Személyes tapasztalatom ebben a programnyelven van a legtöbb. Elöljáróban elmondhatom, hogy csekély mértékben magam is hozzájárultam a jobbá tételéhez, amelyet a jövőben is folytatni kívánok, ezáltal nyílik némi rálátásom a belső működésére is. Amit a PHP-ről *technikai szempontból* érdemes tudni, hogy a 7-es verzióval *rengeteget* gyorsult, számos optimalizálás került megvalósításra, amelyek egy része a *virtuális gép* egyes részeinek az újraírását igényelte. Sajnos emellett azonban küzd néhány, elenyésző számú, de annál bosszantóbb/elképesztőbb *sajátossággal*, amelyek nemcsak a kívülállónak tűnhetnek teljesen érthetetlennek. A fő probléma az, hogy számos ilyen sajátosság a *visszamenőleges kompatibilitás* miatt szerepel benne, és ezek egy részének a megváltoztatását a tagok többsége rendszeresen blokkolja, éppúgy, mint a *statikusabb, típusosabb* irányba történő haladást is (PHP P++ FAQ, 2019). A felhozott, illetve beleértett érvek között a legtöbbször az elterjedtség *mennyiségi csökkenését* lehet megtalálni. Ha azonban valaki megszokja ezeket a *sajátosságokat*, akkor PHP-ban kiválóan *lehet* SOLID rendszerű, nagy teljesítményű, objektum-orientált, magas szintű architektúrájú, átgondolt alkalmazásokat fejleszteni. *Fontos leszögezni, hogy – a Java-val, vagy részben a C#-al ellentétben – a PHP nem kényszeríti a felhasználót az egyébként létező típusosság, az enkapszuláció elveinek, vagy egyáltalán az objektum-orientált szemléletmódnak a használatára*. Itt tehát egy kezdő fejlesztő vélhetően nem fog *magától*, felügyelet nélkül igazodni ezen paradigmákhoz.

A PHP fölött szükség van valamilyen webes keretrendszer használatára. Ebből több, kiforrottnak tekinthető is rendelkezésre áll. Mindegyiknek nagy rajongótábora van, *enterprise* célra a legelterjedtebb a már említett *Symfony*, de nagy népszerűségnek örvend a nyomokban erre épített *Laravel* is. Mindkettővel dolgoztam már, az utóbbinak a fejlesztéséhez szintén igyekszem olykor hozzájárulni – és mindkettővel eltérő problémáim vannak. A Symfonyval szembeni legjelentősebb kifogásom annak köszönhető, hogy összetettsége folytán a teljesítménye csorbát szenvedett, ezért – még a PHP 5-ös verziója alatt – megjelent az a megközelítés, hogy – egyfajta *cache*-ként – az

alkalmazáskonténert *lefordítja*, azaz összeállítja, és eltárolja egészben. Ez olyan problémákhoz vezet, hogy a konténer *statikus*, *nem módosítható* az alkalmazás-kérés életciklusa során. *Természetesen* az alkalmazás-konténer része a *konfiguráció* is, így – a 3.0-s verzióban – nincs lehetőség arra, hogy az inicializálást követően, például egy köztes rétegből (*middleware*), ahol a HTTP fejlécek már elérhetőek, a konfiguráció módosításra kerüljön (hogy a behelyezett vagy módosított értékeket később például a *vezérlőben* futó kód felhasználja). Természetesen számos *work-around*, alternatív megoldás létezik, mint például az eseményeken keresztüli üzenetküldés – csak egyszerűen én annak idején nem értettem, hogy ez *miért volt logikus* az alkotó számára.
A *Laravellel*, amely a legnépszerűbb PHP webes keretrendszer, más jellegű problémák vannak. Én ezzel dolgoztam, dolgozom a legtöbbet, és a fő problémáját abban látom, hogy *alapvető célkitűzése* a fejlesztők számára a – vélelmezett – *egyszerűséget* népszerűsíteni. Emiatt egy csomó minden úgymond *mágikusan*, automatikusan *csak működik*, ami bizonyosan kiváló, amíg a *Pareto elv* szerinti 80%-ba esik, amit szeretnénk megoldani (vagyis *bele van építve* a keretrendszerbe). Ha viszont nem, az felettébb kellemetlen, mert a *mágia* miatt a *hagyományos* megközelítés (vagyis hogy *a kód a legjobb dokumentáció*) itt nem működik: vagy benne van a dokumentációban, amit szeretnénk csinálni, vagy *írt már róla valaki* – vagy ha egyik sem, hát marad a *próbálkozás*..
Természetesen lehet Laravelben *SOLID*, áttekinthető, átgondolt, megtervezett kódot írni. *Amit Laravelben nem lehet, az a csapatmunka, összeegyeztetve az előzőekkel*. Ehhez ugyanis *a Laravel pont azon részeit kell átlépni, amelyek a többieket vonzzák az egyszerűségükkel*. Ez *tökéletesen* megegyezik a *fejlesztői tartományban érzékelt PHP-problémával: ő is PHP (Laravel, ...) fejlesztő, én is PHP (Laravel, ...) fejlesztő vagyok – csak a kettő ég és föld.* És itt jön képbe a *kontraszelekció* – ha (pontosabban *amíg*) valaki *nem érti meg* magától, hogy *mi a probléma* a nem újrahasznosítható kódokkal, a hibás architekturális döntésekkel, addig jó eséllyel annyit fog látni a törekvésből, hogy *az illető most voltaképpen bosszantani akarja – hiszen a Stack Overflow szerint ez így jó, szerinte pedig nem jó.*. A tapasztalatom szerint az ilyesmi végeláthatatlan ellenségeskedéshez és elmérgesedett szituációkhoz vezet; ezeket sajnos – szintén a Pareto elv alapján – statisztikailag a legegyszerűbb a *problémás területek mellőzésével* kezelni.

### *Böngésző- / kliensoldali (front end) technológiák*

Amint az a JavaScript kockázatánál feltárásra került, a *back end* oldallal szemben a *front end* technológiákra *lényegesen* kevésbé jellemző a *kiforrottság*, a *stabilitás*, és az *iparági standardok* használata (vagy egyáltalán *létezése*). Bár a *fejlesztőknél* is tetten érhető a *kimaradástól való félelem (FOMO)*, egy fejlesztő ha rendelkezik is megtakarítással, amely egy esetlegesen megszűnő vagy visszaszoruló technológia helyett az *azt kiszorító konkurrense* elsajátításához szükséges ideig biztosítja a megélhetését, a HackerRank idei felmérése értelmében a fejlesztőknek munkavállaláskor *a második legfontosabb a professzionális karrierépítés-fejlődés* után a *munka-magánélet-egyensúly*a (HackerRank Developer Skills Report, 2019), tehát nem

vélelmezhető, hogy a képzett tartalék felélése, vagy meglévő állás *mellett* technológiai tanulással töltené az idejét. Ezáltal – ha maga nem hordoz *evangélista* jellegeket, amelyek valamely kiforratlan technológia melletti lelkesedéséhez vezetnek – a *marketingesek*, az *evangélisták*, *egyéb üzleti szereplők* jellemzően nem vezetnek technológia-váltáshoz egy feljesztőnél (és bővüléshez is legtöbbször akkor, ha erre a foglalkoztató üzleti szereplő keretet biztosít).

A *böngésző* oldalán a 2000-es évek végén az elsősorban a böngészők inkompatibilitását áthidalni hivatott *jQuery* függvénykönyvtár terjedt el. Ez a mai napig magas népszerűségnek örvend (*a Stack Overflow idei adatai szerint 48 százalékkal az első helyen áll az összes webes keretrendszer között*). Ezt elsősorban az *egyszerűségének* és hibatűrésének köszönheti: számos olyan megoldást alkalmaz, amely *hibaelnyelés (error suppression)* technikán alapul: például ha a megcímzett HTML elem az oldalban nem létezik, akkor *semmi sem történik*, miközben a hagyományos JavaScript-ben *nem definiált változó- vagy null-érték-hiba* is keletkezne, ha ellenőrzés nélkül akarnánk azt módosítani. Bár vitatható (és magam is vitatom), hogy ez a fejlesztő *felelősségérzetének* javítása szempontjából mennyire előremutató, fontos látni, hogy a böngészőkbe épített hibakeresési lehetőségek 2009-es szintje teljes nyugalommal nevezhető *katasztrofálisnak*, és bár napjainkra ezt sikerült egy büszke *közepesen rettenetes* szintre felfejleszteni, ez még mindig viszonylag messze van egy *DOS 6.22* alatt használható *Borland Pascal 7.0* forrásnyelvi nyomkövetőjétől (*debugger*).

A JavaScript fejlődésével (ami nem független a Node.js megjelenésétől) azonban megjelent az igény arra, hogy *front end* oldalon is *átgondolt*, *célorientált*, és *tervezésileg egzakt* módszerek lássanak napvilágot. Két fő irány jelent meg, az egyik a konkrét problémára fókuszáló *UI könyvtárak (library)* létrehozásához, a másik *a minél több mindent oldjunk meg a böngészőben* mottójú, kérdéses koncepciójú mozgalom termékeinek, a *front end keretrendszerek* kialakulásához vezetett.

Természetesen egyik sem mentes a problémáktól. A *UI library* egy konkrét, jól körülírt problémát hivatott megoldani: a felhasználói felület összeállítását és elemeinek kezelését. Ez *elméletben* kiválóan hangzik. A legismertebb képviselője a *React.js*, amelyet a *Facebook* dolgozott ki, és *viszonylag* jól kezeli ezt a problémát. A legalapvetőbb koncepcionális problémája, hogy a tervezője valamiért a *funkcionális programozás* elvét erőlteti (amelynek kétségkívül számos előnye van *egyes tartományokban*); azon belül is azt, hogy legyen minden állapot nélküli, vagyis *stateless.* Eközben viszont *teljesen figyelmen kívül hagyja*, hogy a felhasználói felület (UI) *inherensen, koncepciójából fakadóan tartalmaz állapotot (stateful);* ilyen állapot például a félig beírt, de nem elmentett szöveg; valamely gomb, rádiógomb, jelölőnégyzet stb. állapota, vagy adott esetben valamely mozgatható elem pozíciója. *Tipikus példája* ez annak, amikor két, önmagukban nagy érdemekkel bíró, azonban *vitatható kompatibilitású* paradigmákat kényszerít valaki össze. Emiatt aztán a dokumentációban *külön fejezet van arra*, hogy *ne* használjuk az objektum-orientált paradigmánál megszokott *öröklődést;* helyette a *kompozíciót* erőlteti (React.js Docs). Saját maga még egy *JSX* kódnevű, "XML jellegű JavaScript" szintaxisú

állományformátumot is bevezetett, amely *ismét a szétválasztott felelősség elvének megsértése*, hiszen egy állományon belül keveredik a *futtatható kód* a *felületleíró nyelvvel*. Persze a következő fejezetet ezek miatt annak kellett szentelni, hogy *hogyan gondolkodjunk React módon*.. Szerencsére, bár ez *abszolút nem dokumentált*, erre *a világon semmi szükség nincs*: a React belső működésének megismerését követően *teljesen tökéletesen* működő, kombináltan öröklés- és kompozíció-alapú, objektum-orientált kódot lehet vele írni. A probléma megint ugyanaz ezzel, mint a *Laravelnél* leírtakkal – *egy másik React fejlesztő jó eséllyel nem fogja kapásból átlátni*..

A másik *iskola* célkitűzése ennél jóval vitathatóbb: arra a törekvésre vezethető vissza, hogy *legyen egy karcsú (lean), közös back end szolgáltatás*, korábban *SOAP*, ma jellemzően *ReST*/*JSON* alapon, amely gyakorlatilag csak adatokat ad vissza, ezt pedig a különböző technológiájú *front end* felületek (web, okostelefon-alkalmazás, stb.) *egységesen*, *egyszerűen* fogják majd tudni kezelni.

Tapasztalatom szerint ez a megközelítés, ami gazdasági szempontból *aranybányának* tűnik, egy nagyobb ügyviteli projektben *minden más szempontból a küszöbön álló kataklizma hírnöke*. A hívei azt szokták felhozni, hogy *egy adott szolgáltatásnál (pl. Airbnb, Shopify) ez milyen jól működik.* Nekik igazuk van. Teljesen megfelelő minden olyan feladatra, ahol egy 4-5, nem túl komplex lépésből álló folyamaton kell az érintettet végigvezetni, mint a szállásfoglalás, vagy egy konkrét termék megvásárlása. Némi kompromisszummal alkalmas ezen műveletek kiterjesztett környezetének kezelésére is (város illetve termékkategória kiválasztása után a megjelenő szálláshelyek, vagy éppen termékek viszonylag kis (jellemzően *legfeljebb 20-30*) számának egyidejű letöltésére, lapozgatására. Azonban ha például akárcsak egyetlen helyen is az utolsó 1-2 ezer naplóbejegyzést, vagy rendszereseményt szeretnénk megjeleníteni, kapcsolódó rekordokkal, ott ez a megközelítés *rendkívül magas járulékos terhelést* okozhat.

Ebből az elrendezésből természetesen következik, hogy a *hagyományosan a szerver által elvégzett feladatokat* (pl. hivatkozások / navigáció kezelése, felhasználói felület előállítása, stb.) immár *nem* a szerver végzi, hiszen *sem a SOAP, sem a ReST (sem bármely más, hasonló szerkezet vagy protokoll) erre nem alkalmas* – az ő szerepük pusztán az adat elhelyezkedését és az esetleges szűrési kritériumokat tükrözi, tehát *nem áll túl távol attól funkcionálisan, mintha az adatbázist tudnánk HTTP protokollon keresztül elérni*. Emiatt elkezdték mindezen funkciókat megvalósítani a böngészőben is, kiegészülve az *adatmodell-/állapotkezeléssel*, hiszen az adat immár külön kérésben jött, nem volt *beágyazva az oldalba*. Ez gyakorlatilag odáig vezetett, hogy helyenként a teljes MVC minta – jelentős kompromisszumokkal és félreértésekkel – megjelent a böngészőben. Az első megközelítések (Backbone, Chaplin.js, Ember.js, ..) némelyike még kifejezetten átgondoltnak hatott – cserébe *a front end fejlesztés addigi vélt egyszerűsége esett áldozatul*. Utána jött az AngularJS, amely egyszerűsítette a folyamatokat a fejlesztő irányából – persze ez is a *szétválasztott felelősség elve* megsértésével valósult meg, például a dokumentum gyökéreleme kapta meg a *ng-app* attribútumot (pontosabban ahogy ott nevezik, *direktívát*), ebből tudta a rendszer az adott "alkalmazást" inicializálni. Azóta kijött az AngularJS második verziója, ami csak

Angular 2 névre hallgatott. Azóta az Angular elérte a 6-os verziót, sokat fejlődött, az egyszerűsítést, mint célkitűzést csak korlátozottan érte el (), viszont a szétválasztott felelősség kérdését azóta sem sikerült maradéktalanul rendezni. Ráadásul, mint a JavaScript kockázatnál kitértem rá, *szokást csinált* abból, hogy a *HTML*-re, mint kvázi (*néhai*) szabványra rákényszeríti a különféle *nem konvencionális* ötleteit..

# 4. HOZZÁÁLLÁS ÉS TELJESÍTMÉNY VIZSGÁLATA

## 4.1. Hipotézisek

A kutatás első része arra keresi a választ, hogy mekkora szerepet játszik a *belső*, illetve a *külső* kockázatok külön-külön, az érintettek által vélelmezett mértéke abban, hogy a hazai mikro-, kis- és középvállalkozások IT-támogatott ügyviteli integrációjának mértéke alacsony.

A feltárt és valószínűsíthető kockázatokat *két csoportra* szükséges bontani.

Az első csoportba sorolhatóak azok a kockázatok, amelyek *a termék fejlesztéséhez tartoznak*, és amelyekre a fejlesztőnek, a projektgazdának *befolyása van* (*belső kockázatok*). Ide sorolandó a termékkockázat, a projektkockázat, valamint a technológiai kockázat is (hiszen ez elsősorban a választott technológiákon, és a fejlesztési módszertanon, ezek hatékonyságán múlik).

A második csoportba sorolhatóak azok a kockázatok, amelyek *a projekt szempontjából externáliának* tekinthetőek, ilyenek az ügyfeleknél felmerülő kockázatok többsége: például a *konjunkturális kockázat*, valamint az érintett szereplőkkel szemben esetlegesen fennálló *bizalmatlanság.* Ezeket nem lehet a fejlesztés paraméteréül szabni, hiszen nincs befolyásom rájuk.

A kutatás második felében megvizsgálom a rendelkezésre álló *tervezési mintákat* és *webes technológiákat*, trendeket, elsősorban teljesítmény-szempontból. Itt arra keresem a választ, hogy vajon ténylegesen a legelterjedtebb, legnépszerűbb megközelítések segítik-e a legjobban a *strukturálisan fenntartható* fejlesztést.

**A kutatás hipotézisei**

- A *célcsoport* IT-támogatott ügyviteli integrációs fokának *alacsony* szintjében a *belső kockázatok* szerepe *elhanyagolható*.

- A *célcsoport* IT-támogatott ügyviteli integrációs fokának *alacsony* szintjében a *külső kockázatok* szerepe *elhanyagolható*.

- A rendelkezésre álló *legújabb* fejlesztési, tervezési módszertanok és technológiák, valamint az ezek *fejlesztői* által javasolt *best practice-ek együttesen garantálják* egy magas *fajlagos teljesítményű*, webes felületű, üzleti-ügyviteli célú alkalmazás rövid idő alatti, *alacsony kockázatú* fejlesztését.

*A tervezettek szerint az első két hipotézis vizsgálata statisztikai módszerekkel, míg a harmadiké empíriával történik.*

## 4.2. Az érintettek érdekeltségének és elvárásainak primer elemzése

### 4.2.1. Az *érdekeltek* azonosítása

A fejlesztéshez, bevezetéshez kapcsolódó projekt- és termékkockázatok lehetőség szerinti minimalizálásához éppen úgy, mint a kérdőív célcsoportjainak azonosításához szükséges az érdekeltek szempontjainak összegyűjtése és elemzése, és mind a termékkel/projekttel szembeni kezdeti elvárások, mind a felelősségi körök *előzetes* tisztázása.

Ehhez elsőként elengedhetetlen az érdekeltek (stakeholder-ek) számbavétele.

**Stakeholder-ek**

- **Ügyfél: döntéshozó**
- **Ügyfél: tényleges használó**
- **Fejlesztő, integrátor: döntéshozó**
- **Fejlesztő, integrátor: kivitelező** (aki a tényleges fejlesztést, integrációt végzi)
- **Projektgazda / Projektmenedzser** (fejlesztő/integrátor általi vagy független)
- **Kormányzat**
- **Szponzor (támogatás esetén)**
- **Infrastruktúra-szolgáltató**

Természetesen ezen felül számos szereplőt lehetne még a folyamatba bevonni, azonban itt ezen analízis elsődleges célja a fejlesztés bemenő kritériumainak meghatározása, így az arra előzetes hatással nem bíró szereplőket mellőztem.

A *4. ábrán* látható a fenti szereplők Engel-Salomon karakterisztika-táblázata.
Ehhez csak egyetlen ponton fűznék magyarázatot: a projektsikerességre vonatkozó esorolásban a kormányzat hatása (is) megelőzi a döntéshozókat.

Ennek az az oka, hogy álláspontom szerint megfelelő fejlesztési módszertan és projektmenedzsment választása esetén viszonylag kevés olyan, előre abszolút nem látható, később, felbukkanásukkor nehezen vagy egyáltalán nem orvosolandó nehézséggel lehet számolni, amely akár az ügyfél, akár a fejlesztő érdekkörében keletkezne, ugyanakkor egy volatilis jogi-szabályozási környezetben (e pillanatban ilyennek tekinteném az egész EU-s makrokörnyezetet, nemcsak a hazait) bármikor bekövetkezhet olyan szabályozási változás, amely valamely már befejezett, de még nem átadott, vagy akár használatba is vett fejlesztést megfoszt a hasznosságától, és *ezt a kockázatot nem lehet az esetleges későbbi ügyfeleken szétteríteni*.Az így meghatározott sorrend alapján egyszerű az érintettek érdek-befolyás mátrixban történő elhelyezése; ez az *5. ábrán* látható.

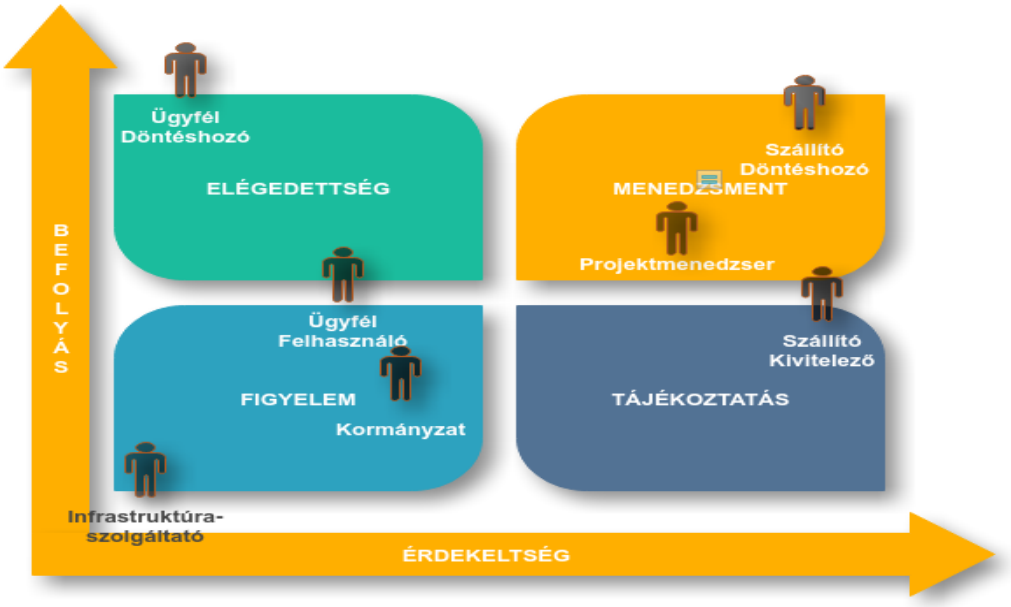
**Stakeholder-ek tulajdonságai, befolyásuk a döntéshozatalra és a bevezetés sikerességére**

| Stakeholder | Elsődleges motiváció | Pozíció *(befolyás a döntéshozatalra)* | Hatás *(hozzájárulás a projekt sikeréhez)* |
|---|---|---|---|
| Ügyfél: döntéshozó | Idő- és költségmegtakarítás | 1 | 6 |
| Ügyfél: felhasználó | Frusztráció és hibalehetőségek visszaszorítása | 4 | 3 |
| Fejlesztő, integrátor: döntéshozó | Meglévő funkciók értékesítése egyedi fejlesztés helyett | 2 | 7 |
| Fejlesztő, integrátor: kivitelező | Konzisztens, konzekvens bejövő igények | 5 | 2 |
| Kormányzat | Megfeleltetés jogszabályoknak és standardeknek | 6 | 5 |
| Infrastruktúra-szolgáltató | Munkaóra-arányos bevétel maximalizálása | 7 | 4 |
| Projektgazda / Projektmenedzser | Ügyfél- és fejlesztőérintettség maximalizálása | 3 | 1 |

*4. ábra: Az érintettek Engel-Salomon karakterisztika-táblázata*

*Saját grafika*

**A fejlesztési projekt szereplőinek elhelyezkedése az érdekeltség-befolyás mátrixban**




*5. ábra: Az érintettek érdekeltség-befolyás-mátrixa*

*Saját grafika*

Ez a szempont azért fontos, mert később egyes nem kezdeti funkciók fejlesztése történhet már egy-egy adott ügyfél konkrét igényére, vagy – az arról való értesülést követően – megfigyelése mellett, ahol már az esetleges ügyfélmegtartásra is szükséges lehet törekedni a projektmenedzsment tárgykörébe eső kommunikáció során (függetlenül attól, hogy az adott ügyfél esetlegesen igényli-e, megrendeli-e majd az elkészült funkciót, vagy sem).

*A karakterisztika-mátrix megerősíti, hogy az ügyfél, megrendelő oldaláról a döntéshozó és a felhasználó is célcsoportot kell képezzen.*

### 4.2.2. Szempontok, elvárások

A következő lépésben összegyűjtöttem az érintettek összes, általam előzetesen valószínűsíthető elvárásait és szempontjait *úgy a turnkey SaaS bevezetése, mint egy vagy több esetleges egyedi funkció későbbi megrendelése esetére*.
Ennek célja a közöttük esetlegesen fennálló ellenérdekeltségek feltárása, amelyek előzetes állásfoglalást, vagy stratégiai döntést igényelnek a fejlesztés adott szakaszának megkezdése előtt.

Az azonosított *érdekeltek szempontjait, elvárásait az I. mellékletben foglaltam össze.*

## 4.3. Tervezési minták és technológiák kiválasztása

### 4.3.1. Infrastruktúra, háttértechnológiák

Az iparági gyakorlatnak és saját tapasztalatomnak megfelelően alapvetésként kezelem, hogy az infrastruktúra *nyílt forráskódú komponensekből* álljon, bemeneti feltétel tehát a Linux operációs rendszeren történő futtathatóság.

Figyelembe véve a C# linuxos változatának, a Mono projektnek a helyzetét; azt a tényt, hogy Java servlet-ek biztonságos üzemeltetésében szerzett tapasztalatokkal egyáltalán nem rendelkezem; illetve a technológiai áttekintésnél feltárt egyéb érveket és ellenérveket, a platformok közül az ott feltárt okok miatt a PHP-ra esett a választásom.

Adatbázis-szerverként a MySQL családot részesítem előnyben. Bár az alapprogram szintén *nem mentes* az Oracle-kockázattól, a nyílt forráskód miatt több különböző piaci szereplő épített az alapcsomagra *saját változatú terméket*, önálló funkciókkal, és támogatással (Percona, MariaDB). Ez – szemben a Java-val, ahol a legfőbb *felhasználó* Google ismét inkább a *Kotlin* nevű programnyelvet *erőlteti* Androidon – megalapozza azon vélelmem, hogy a projekt nem fog a közeljövőben *leállni*. Igaz, az Oracle nem is *töri magát* a funkcionalitás bővítésével (személyes meggyőződésem, hogy *ha tehetné*, inkább rég becsukta volna a projektet, hogy a saját névadó termékének a piacát növelje), megteszi ezt azonban a MySQL eredeti alkotója, aki a MariaDB mögött áll.

Felmerült a ma *divatos trendeknek* megfelelően, hogy valamely nem-relációs (*közismertebb, de pontatlan kifejezéssel NoSQL*) adatbázist alkalmazzam. Azonban – amellett, hogy *természetesen elismerem, hogy nem minden adathalmaz tárolható vagy ábrázolható relációk mentén* – személyes véleményem szerint az *adatmodellezéstől, majd annak normalizálásától való kényszeres elzárkózás* a legújabb idők fejlesztői *kontraszelekciójának* az egyik tünete.
Ennél jóval fontosabb viszont, hogy például a bolygó második legnagyobb webszerkesztővel egybekötött *ingyenes tárhely*-szolgáltatója, a WIX *vezető architektje* tapasztalatai szerint nagy adatmennyiség és magas skálázhatósági igény mellett (százmilliós nagyságrendű rekordról van szó) *"a MySQL egy jobb NoSQL"* (ABRAHAMI, Y., 2016).

Ennek megfelelően a MariaDB jelenleg 10.3 változata várhatóan a jelen és a jövő kihívásaira is megfelelő válasz lesz, nem modellezhető adatok esetén is.

Emellett hangsúlyt fektetek arra, hogy ahol SQL lekérdezés írása válik szükségessé, ott az *ANSI SQL* szintaxist részesítsem előnyben (különös tekintettel a MySQL-specifikus karakterlánc-határoló *backtick ( ` ) karakter használatának mellőzésére*). Ezeket egyébként egy inicializációs paranccsal be lehet állítani kapcsolódáskor. Ezáltal lehetővé válik az adatbázis-réteg egy másik, szabad alternatívára történő cseréje (pl. *PostgreSQL*), illetve egy esetleges *kihelyezett (on-premise)* igény esetén a már rendelkezésre álló és nem lecserélhető adatbázis-technológia használatára.

### 4.3.2. Kiszolgáló-oldali technológiák

A back end technológiák áttekintését követően az ott taglalt okok, és a tapasztalataim alapján a fejlesztéshez a *saját keretrendszerem* használom.

Ennek legfőbb oka, hogy mivel belső használatra készült, nem akar *több lenni a szükségesnél*, különféle elvárásoknak megfelelni, ugyanakkor *nem a (többi keretrendszer) megtapasztalásának elkerülése* céljából kerül használatra (és egyáltalán fejlesztésre), hanem elsősorban éppen ezen tapasztalatok miatt. Ez egy fontos különbség, amely ugyanakkor egy kívülálló számára nem szükségszerűen szembetűnő. *Természetesen* ennek is megvan a kockázata: *internetes dokumentáció* nem lévén biztosan nem fogok tudni olyan, *a kódot nem értő*, sablonszerűen dolgozó *"junior"* fejlesztőt bevonni, akik gyakorlatilag minden olyan technológiánál, amelyhez van *online*, pl. *YouTube-n* hozzáférhető oktatóanyag, rendelkezésre állnak. (Hogy ez *valójában milyen jellegű* kockázatot jelent, elsősorban *kinek* és *milyen mértékben*, arra a keret szűkössége miatt itt nem térek ki.)

### 4.3.3. Böngésző-oldali technológiák

A *front end* terén a React-vonalon szándékozok elindulni.

Ennek fő oka, hogy itt ez az a megközelítés, amely *nem akar több lenni* annál, ami. Sajnos a React immár maga is egy *igen erőforrás-igényes* halmazzá nőtte ki magát, van neki azonban folyamatosan *több párhuzamos* alternatívája, amelyek fő érvként jellemzően a jóval alacsonyabb hely- és teljesítmény-igényt hozzák fel maguk mellett. Emellett változatos további célokat tűznek ki, *ezek közé nem tartozik feltétlenül a teljes kompatibilitásra való törekvés*.

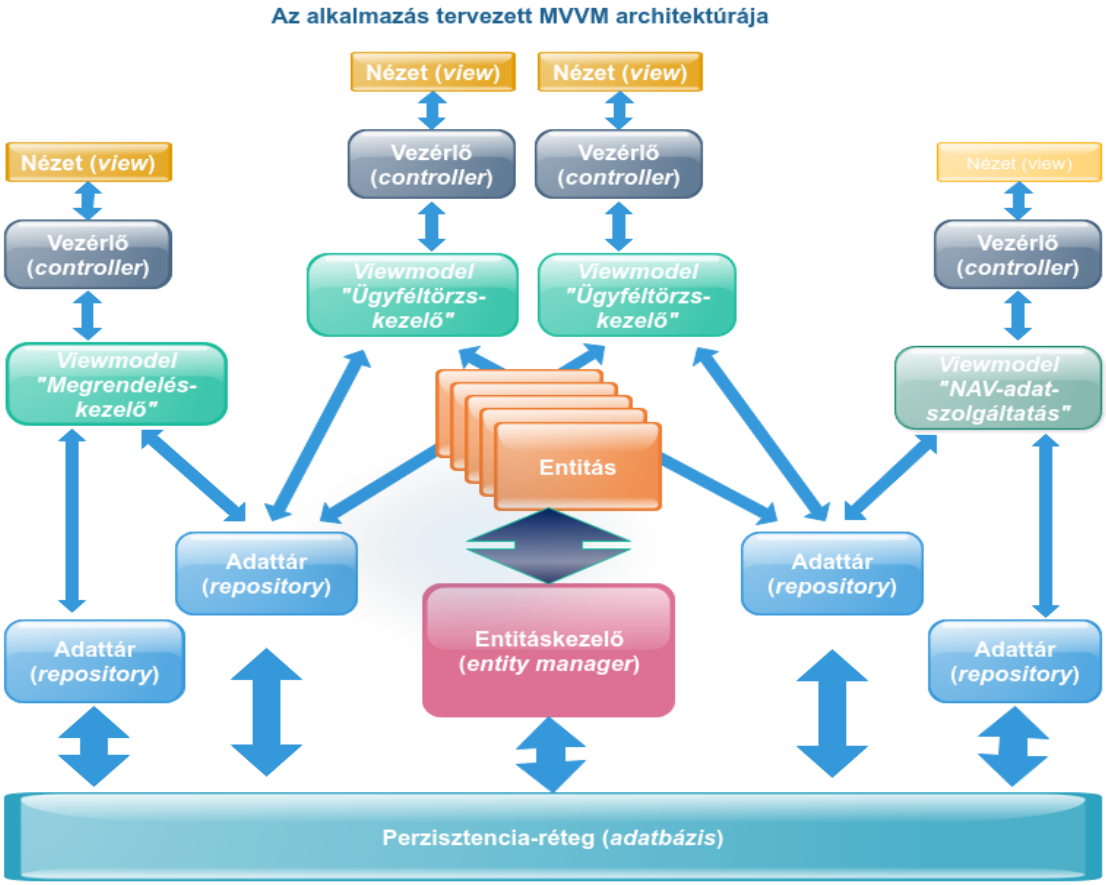
A *Preact* nevezetű csomag ezek közül a valamikor *3 kbyte* tömörített méret mellé *viszonylagos kompatibilitást* is biztosít, törekszik a teljes kompatibilitásra, és emellett folyamatosan fejlesztés alatt áll.

Emellé a Javascript böngészőkben fellelhető nyelvi változat, az *ES5.1* dialektusát alkalmazom. Ez nagyjából az összes, jelenleg még legalább biztonsági frissítésben részesülő böngészőben működik (IE8-ban például viszont nem). *Ez nem jelenti azt, hogy nem kellene ismernem a JS újabb szintaxisait, vagy akár a React-féle JSX formátumot – hiszen a komponensek legnagyobb része ezeket használja, így bármely*

*azokat érintő módosítás igényli az alkalmazott dialektusok ismeretét.* Azt ugyanakkor igen, hogy ha *idővel* kiderülne, hogy valamelyik React-megközelítés *elérte a falat* a triviális teljesítmény-optimizálás során, akkor lehetőségem nyílik egy *egyedi rendering (megjelenítő)* megoldás létrehozására, amely – nyilván magasabb menedzsment-igény mellett – gyorsabb teljesítményt biztosíthat anélkül, hogy le kellene cserélnem a *jól bevált* vagy *megszokott* UI-komponenseket.

### 4.3.4. Tervezési minták

Az architekturális minták közül az *MVC* módosított változata, az *MVVM* minden olyan funkciót megvalósít, amelyre a rendszer szerkezete alapján igény van, miközben a SOLID irányelvek szerinti elhatárolásokat biztosítja. Ez természetesen nem jelenti azt, hogy *nincsenek* vezérlők: a beérkező kérések *feldolgozását* továbbra is a vezérlő *cselekménymetódusai (action)* végzik, azonban az adatok kívánt formátumba alakítását a *viewmodel* látja el. Emellett az adatkapcsolati réteg semlegességének, illetve a nem *entitásszintű* lekérdezéseknek a támogatásához az *adattár* (repository) mintát is fel kívánom használni.




*6. ábra: Az üzleti alkalmazás MVVM architektúrájának blokkvázlata*
*Saját grafika*

A *6. ábrán* látható az alkalmazás tervezett architektúrájának blokkdiagramja.

### 4.3.5. Felhasznált technológiák teljesítmény-benchmarking alapú összehasonlítása

A *komparatív (összehasonlító)* teljesítmény-benchmarking alapja *a megfelelően körülírt teszt-eset*. Bár a weben rengeteg *benchmark*-eredmény fellelhető szinte bármilyen platformról és technológiáról, ezek többsége *tudományos szemszögből értelmezhetetlen*. Ennek oka többféle. Nagyon gyakori, hogy *almát hasonlít össze körtével*, vagyis a környezet kialakítása nem standardizált, az összehasonlítás teljes mértékben *megalapozatlan*. A leggyakoribb hiba mégsem ez, hanem *a vizsgált teszt-eset teljes irrelevanciája*. A legtöbbször egy webes platform esetén egy *demó jellegű, üres, vagy "Hello, World" összetettségű választ kiszolgáló alkalmazás "mérése" szokott előfordulni* – talán fölösleges kifejteni, hogy mi ezzel a probléma, vagy mennyire tükrözi ez a *teszt-eset* egy valódi *üzleti-ügyviteli* vagy akár *kommunikációs* alkalmazás működését. Ezek mellett ügyelni kell arra is, hogy *dinamikus adatok feldolgozására* kerüljön sor a teszt-esetben – egyébként hiába a *reprezentatív modell*, az esetlegesen beépített *gyorsítótárazási (caching) mechanizmusok* miatt már a második kérés kiszolgálási ideje is torzítani fog *statikus* adatok mellett – és az összes ilyen cache-elési mechanizmusnak az észlelése is gondot okozhat, a paraméterezésük, ki-bekapcsolásuk, ürítésük módszereivel, nehézségeivel ha egy könyvet nem is, egy fejezetet biztosan meg lehetne tölteni.

Az összehasonlításban *Java környezetből a Spring 4 / SpringBoot (csak ORM), PHP környezetből pedig a Symfony 5.0.1 (Query Builder és ORM), a Laravel 5.8 (Query Builder és ORM), valamint a saját keretrendszerem (nyers SQL)* kerül mérésre. A legérdekesebbnek számomra *előzetesen* az objektum-reláció-leképezés (ORM) és a lekérdezés-generátor (Query Builder; gyakorlatilag objektum-orientáltan teszi lehetővé az SQL lekérdezés karakterláncként történő összeállítását) között várható teljesítmény-különbség ígérkezik.

A benchmarking környezet és a teszt-eset karakterizálása a *IV. mellékletben* található.

# 5. A KUTATÁS EREDMÉNYEI

## 5.1. Irodalmi tanulságok

A szekunder kutatás során feltárt adatoknak a deduktív kiértékelését részesítem előnyben; ez két *közvetlen* következtetéshez vezet.

### *Csapatszellem és technológiai kockázat*

Az első, legnagyobb technológiai tanulság a csapatszellem és a technológiai kockázat összefüggéséből vezethető le: *észrevehető*, hogy az agglomerálódó elven növekvő, *monopóliumszerűen* működő piaci szereplők sokszor *öncélú*, de legalábbis a külső érintettek számára legfeljebb *korlátozottan indokolható* belső *döntési folyamatai* ugyanolyan károsnak és kockázatosnak tekinthetőek a technológiát igénybe vevő, hasznosító szereplők szemszögéből, mint a *hosszú folyamatok során feltárt problémákat* az azokra nyújtott, *bevált* megoldásokkal együtt figyelmen kívül hagyó, *guru* jelleggel népszerűsített *egyedi*, *szigetszerű* törekvések.

Feltárásra került az is, hogy miközben a *tervezési minták* mellőzése, vagy éppen racionális alkalmazhatósági vizsgálat nélküli, *futószalag-szerű* alkalmazása szintén nem elhanyagolható gondot jelent, a mai munkaerő-hiányos környezetben a hiány enyhítésére kidolgozott *pótmegoldások egyike sem* mutat abba az irányba, hogy ezt a kockázatot csökkenteni tudná.

Mindezek értelmében megállapítható, hogy a PwC kérdőíves vezetői kutatásainak eredményei közül a *technológiai, az ezzel összefüggő munkaerő-piaci*, illetve *know-how* alapú *aggodalmait* az elmúlt *két évtized* áttekintett kutatási, üzleti tapasztalatai *közvetlenül, vagy közvetett módon teljes mértékben alátámasztják;* arra pedig, hogy valamely aggasztó trend terén pozitív irányú *trendforduló* lenne kilátásban, semmilyen jel nem mutat. A kérdőívben is szereplő *konjunkturális* aggodalmakat ugyanakkor jelen értekezés keretein belül csak az érintett területekhez történő kapcsolódás mentén vizsgáltam, így arról *megalapozott* képet alkotni nem állhat szándékomban.

### *Virtualizáció, felhőszolgáltatások*

Az adott feltörekvő technológiák Gartner által kidolgozott *hype ciklus* szerinti *állását a Gartner minden évben közreadja, és azon szerepel a legtöbb technológia, érdeklődési adatokkal alátámasztva. A grafikonról azonban nemes egyszerűséggel lemaradt a virtualizáció, illetve a felhő jellegű hosting egésze.* Tekintve, hogy a ciklus *csúcsának meghaladása nem* jelent teljes meghiúsulást, emellett olyan, szintén feltörekvő és absztrakt technológia *is* szerepel rajta, mint *a mesterséges intelligencia*, erre *semmiféle objektív magyarázatot* nem sikerült fellelnem. Az irodalmi adatok alapján, illeve a bemutatott, az *Amazon* árbevételének alakulását mutató ábra alapján azonban *egyértelmű*, hogy – kivéve azt az esetet, ha valamely feltáratlan okból *a Gartner cikluselmélete nem volna vonatkoztatható ezen technológiára* – a virtualizáció, illetve a nyilvános felhő jelenleg az "*eltúlzott várakozások csúcsa" fázist megelőzően, a*

*növekedés állapotában van*, és (amint ez ebben az állapotban *bármelyik* valós idejű pillanatban érvényes) *sosem volt még ilyen közel* a nevezett csúcs eléréséhez.

A bevezetőben említett *DESI* indexből is a mögötte rejlő EU-munkacsoportok azon álláspontja érhető tetten, hogy *a feltüntetett értékek* – így a jelen dolgozat szempontjából legfontosabb 4. *"A digitális technológia integrációja"* dimenzió is – 100%-nak az *optimális, elérendő* szintet tekinti. Ez azon vállalatok százalékában értelmezendő, akik *igénybe vesznek, használnak egy adott technológiát;* így például a *4a4*. szerinti *közepes-magas összetettségű felhős szolgáltatást (PaaS, SaaS, IaaS).* Véleményem szerint a *SaaS* (*szoftver, mint szolgáltatás*) *összemosása* mind a *PaaS* (*platform, mint szolgáltatás*), mind az *IaaS* (*infrastruktúra, mint szolgáltatás*) *kontraproduktív* mind üzleti, mind szakmai szempontból; ugyanis a SaaS futhat privát felhőben, és mint ilyen, *számottevően kevesebb kockázatot hordoz önmagában*, mint akár az *IaaS*, akár *a PaaS*, melyek *lényegükből fakadóan (inherensen) nyilvános felhőnek* tekintendőek.

A rendelkezésre álló adatokból ugyanis kiderül, hogy *gyakorlatilag az összes, ma felhőszolgáltatáshoz használt processzor olyan sebezhetőséggel küzd, amely lehetővé teszi valamely virtuális gép (bérlő, program, stb) számára, hogy egy elvileg izolált másik virtuális gép (bérlő, program, stb.) vagy akár a gazdagép memóriájához hozzáférjen;* emellett ez a sebezhetőség *egyedül olyan módon* lenne *javítható*, amely a számítógépek, kiszolgálók teljesítményét *nagyjából a 2002-2005-ös szintre* vetné vissza. Ez akkora probléma lenne, hogy *a közeljövőben sem várható a megjelenő processzorokban a sebezhetőtől (spekulatív végrehajtás) eltérő technológia alkalmazása.*

Az MNB *felhő-ajánlása* számos, a szolgáltatást biztosító üzleti szereplők érdekével ellentétes, ugyanakkor az igénybe vevők érdekeinek, továbbá biztonsági, valamint *compliance* szempontoknak kedvező *ajánlást* (kvázi feltételt) tesz, elsősorban a *kritikus szolgáltatások* céljára(Magyar Nemzeti Bank, 2019). Az *itt felsorolt* kockázatokra azonban a legkevésbé sem tér ki; pusztán az *előny-hátrány elemzést* írja elő, amelyre csak *minimumkövetelményeket* ad meg. *A további vizsgálandó szempontok meghatározásánál a felhő kockázatainál leírtak értékes támpontot adhatnak az érintetteknek.*

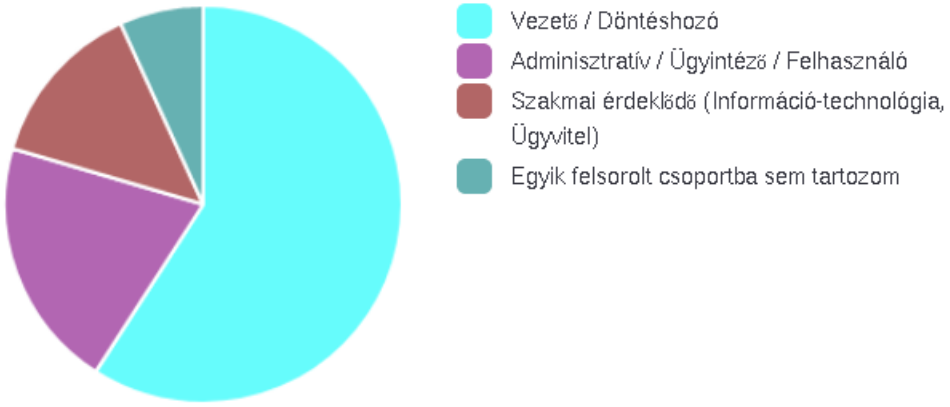
Mindezen kockázatok lényegesen csökkenthetőek, vagy a megfelelő információ-biztonsági elvek betartásával nagyjából meg is szüntethetőek *saját üzemeltetésű (on-premise) kiszolgáló vagy akár felhő alkalmazásával.*

Mindezekből kifolyólag az EU álláspontja a nyilvános felhő alapú szolgáltatások használata, mint kívánalom terén *enyhén szólva aggályosnak* tekinthető.

## 5.2. A válaszadói adatok kiértékelése

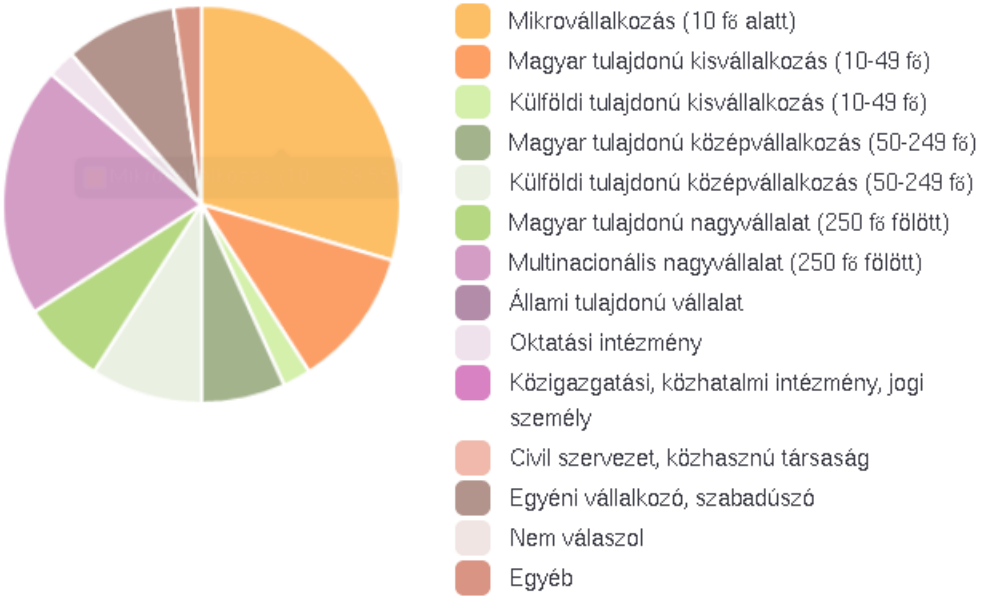
### 5.2.1. Demográfiai áttekintés

A kérdőívet összesen 61 fő töltötte ki. Ebből két kitöltő válaszai teljességgel értékelhetetlenek, míg további négy fő a *szoftverszempontok fontossága* résznél felhagyott a kitöltéssel, ezáltal kvantitatívan értékelhető adatokkal nem szolgáltak.




*7. ábra: A megkérdezettek érintettség szerinti megoszlása*
*Saját grafika*

A válaszadók többsége (71%) *férfi*, 59% *vezető/döntéshozó* szerepben tevékenykedik, 64%-uk 30 és 39 év közötti, és háromnegyedük (75%) budapesti lakos. Felülreprezentáltak közöttük a diplomások: 75% rendelkezik főiskolai vagy egyetemi alap- vagy mesterfokozattal.




*8. ábra: A megkérdezettek foglalkoztató szerinti megoszlása*
*Saját grafika*

A célcsoportba (*egyéni vállalkozó, mikro-/kis- és középvállalkozás)* 68% sorolható, míg 27% hazai vagy külföldi tulajdonú nagyvállalat alkalmazottja. Tekintve, hogy egyebek

mellett a Világbank és az EU 2018-as adatai szerint az Egyesült Államok *kockázatitőke-befektetőinek (business angel) 45 százaléka corporate területről érkezik* (The World Bank Group, 2018, p. 109.), az ő véleményüket *is* célszerűtlennek tartanám figyelmen kívül hagyni.

A demográfiai adatok értékelésénél *meg kell vizsgálni* annak várható esélyét is, hogy milyen irányban torzíthatja az eredményeket a *nem-reprezentatív* mintavétel. Az *EU Nem-egyenlőségi Intézet* 2018. év végi tanulmányából kiderül, hogy az EU-28-on belül a *férfiak digitális magabiztossága* 10 százalékkal (73 vs. 63), a *digitális készsége* egy százalékkal (93 vs. 92) magasabb, míg a *digitális oktatásban való részvételük 83-17 (százalék) arányban oszlik meg, a férfiak javára* (European Institute for Gender Equality, 2018). Emellett az EU Európaipolitika-kutató Központ által támogatott 2019. év eleji kutatás szerint *továbbra is fennáll egy város-vidék ellentét a digitalizáció, az Internet-hozzáférés;* továbbá *hazánk esetében a fővárosi és közép-magyarországi régióban a reprezentatívnak tekintett elektronikus közigazgatási kapcsolattartás szintje számottevően meghaladja a többi régió szintjét* – kiemelve, hogy *a technológiai növekedés túlnyomórészt város-szemléletű és -központú, és annak szándékolt növekedésében az oktatás központi szerepet játszik* (VIRONEN, H., KAH, S., 2019).

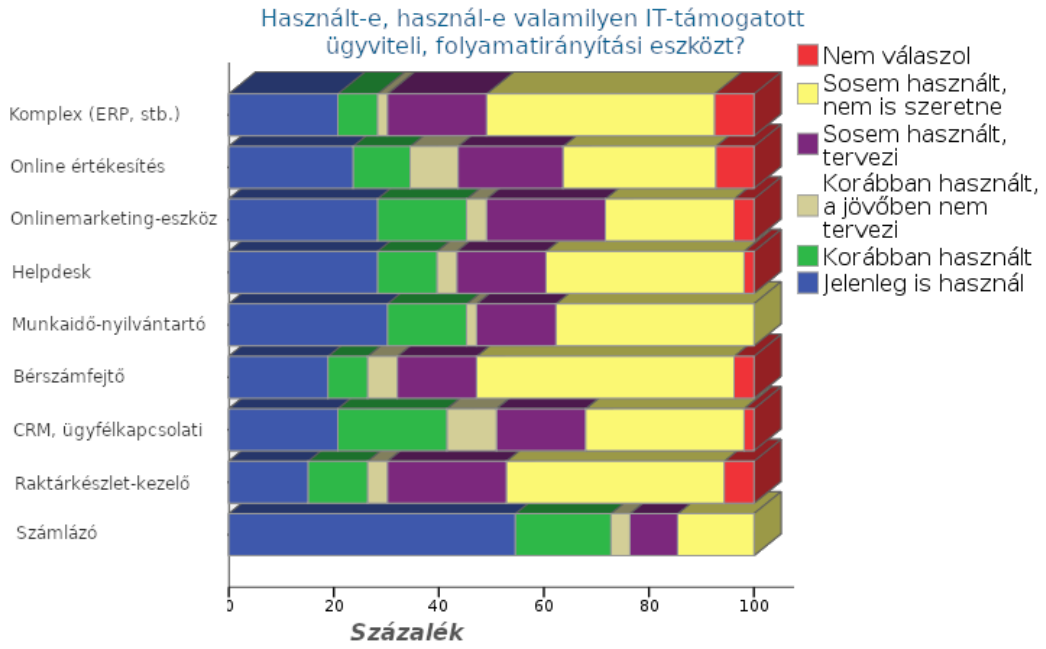
Mindezek alapján *nem várható*, hogy az érintett döntéshozók és/vagy felhasználók *nem*, *lakóhely* vagy akár *képzettség* szerinti reprezentatívabb elérése a *lelkesedés* irányába mozdítaná az eredményeket (az ellenkezője viszont joggal feltételezhető). Vagyis a vizsgált minta szentimentuma az irodalmi adatok alapján *lelkesnek* feltételezendő. Ez számomra nem zavaró, hiszen ők az optimális célpiaca egy *az ő igényeiket kielégítő* e-ügyviteli rendszernek – azonban az eredmények *extrapolálásánál* ezt figyelembe *kell* venni.

Az adatok kiértékelése során érdekes volt megfigyelni, hogy a válaszokból nem volt levonható olyan ellentmondás, amely a kiegészítő, azaz a kutatási kérdésekkel közvetlen kapcsolatba nem hozható kérdésekre adott válaszokkal való korreláció vizsgálatát indokolta volna, ezért *ezen kérdések adatainak elemzésére, illetve ilyen korreláció-vizsgálatra nem került sor*. Az egyetlen vélelmezhető ellentmondás a *felhő-szolgáltatásokkal* kapcsolatban látott napvilágot, ennek lehetséges okaira külön *kitérek*.

### 5.2.2. Előzetes tapasztalatok, elvárások a termékekkel kapcsolatban

Előzetes tapasztalat, hogy *egyetlen válaszlehetőség megadása sem volt értelmetlen* – egy kivétellel a *"korábban használtam, de a jövőben nem tervezem"* tartalmú, trendszinten esetleg értelmetlennek tűnő válaszlehetőség is szerepel; van, ahol nagyjából kilenc százalék választotta ezt.

A megkérdezettek többségének van tapasztalata az IT-támogatott ügyviteli, folymatirányítási megoldások terén: összességében 45 százalék mondta azt, hogy korábban már használta valamely terméket; emellett 34 százalék tervezi használatát a jövőben.

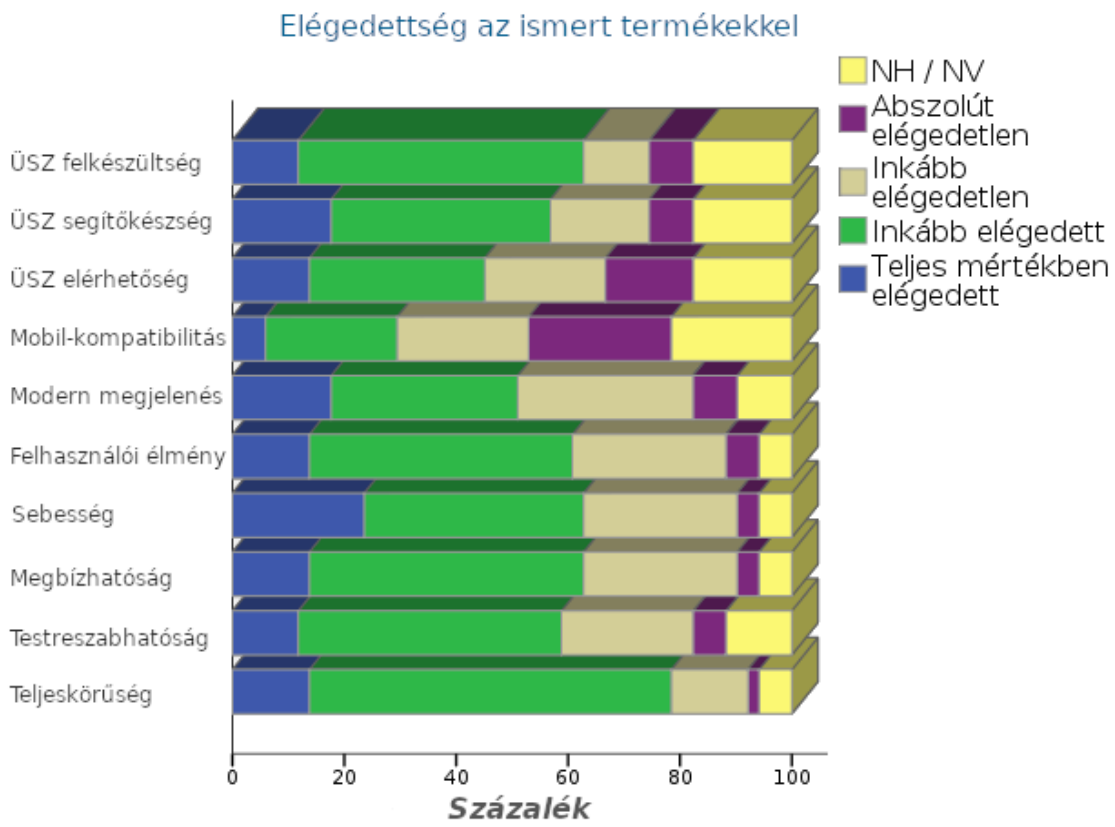



*9. ábra: IT ügyviteli megoldással kapcsolatos tapasztalatok*
*Saját grafika*

A válaszadók többsége a felsorolt szempontok többségével *inkább vagy teljes mértékben elégedett* a korábban vagy jelenleg használt termékek esetében. Ez azt jelenti, hogy *nem elrugaszkodott* gondolat a meglévő elvárásaiknak megfelelni egy új termékkel.

A leginkább elégedettek a megkérdezettek a *funkciók teljeskörűségével*, de a sebesség és a felhasználói élmény is a legjobb eredmények közé tartozik. A *legkritikusabbnak* a mobileszközökön való használhatóság bizonyult, amelyet közvetlenül az *ügyfélszolgálat elérhetősége, hozzáférhetősége* követ.

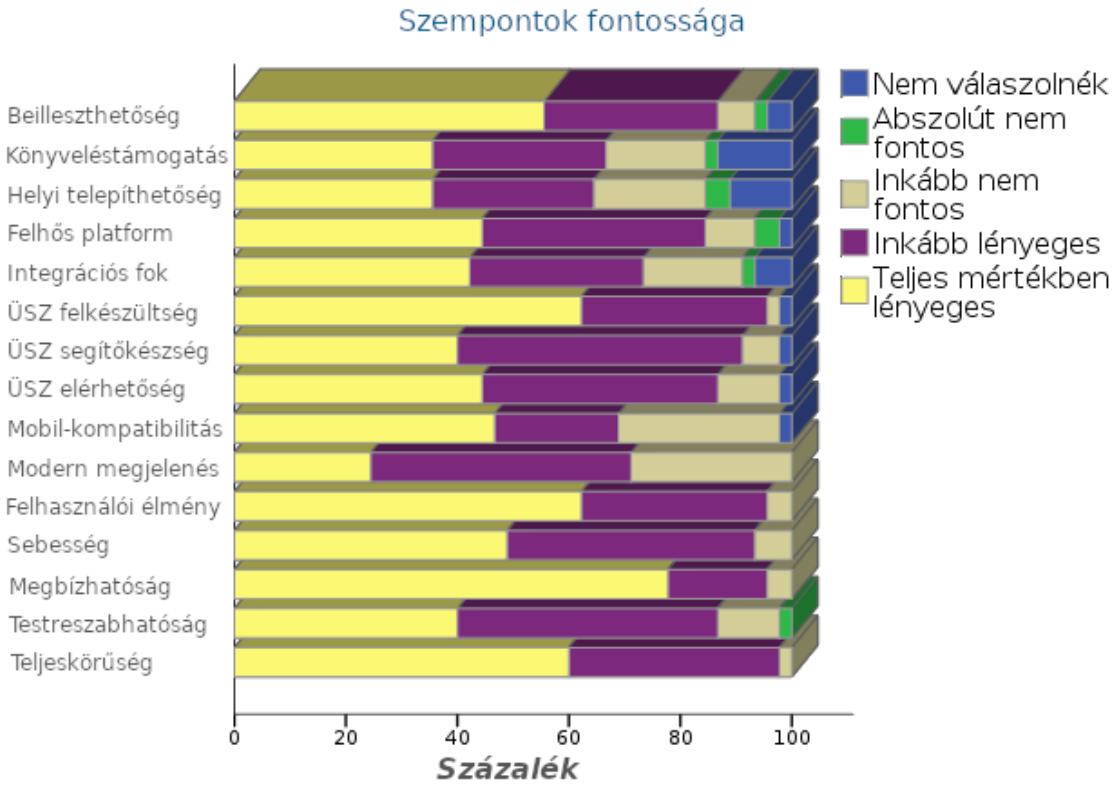
Egy kissé más képet mutat az adott szempontok *szubjektív fontossága.* Itt több érdekes tanulság is levonható. Az érintettek *az összes felsorolt szempontot fontosnak* ítélték. Összesítve a legfontosabb sorrendben a *teljeskörűség*, az *ügyfélszolgálat felkészültsége* és a *megbízhatóság* lett – ez részben egybeesik azzal, hogy *teljes mértékben lényegesnek* sorban a *megbízhatóság*, a *felhasználói élmény*, és az *ügyfélszolgálat felkészültsége* számít. A legkevésbé kritikusak a *helyi telepíthetőség*, a *könyvelés/mérlegkészítés támogatása*, a *mobileszközökön való használhatóság*, és a *modern megjelenés* lett.




*10. ábra: Korábban megismert termékekkel való elégedettség*

*Saját grafika*

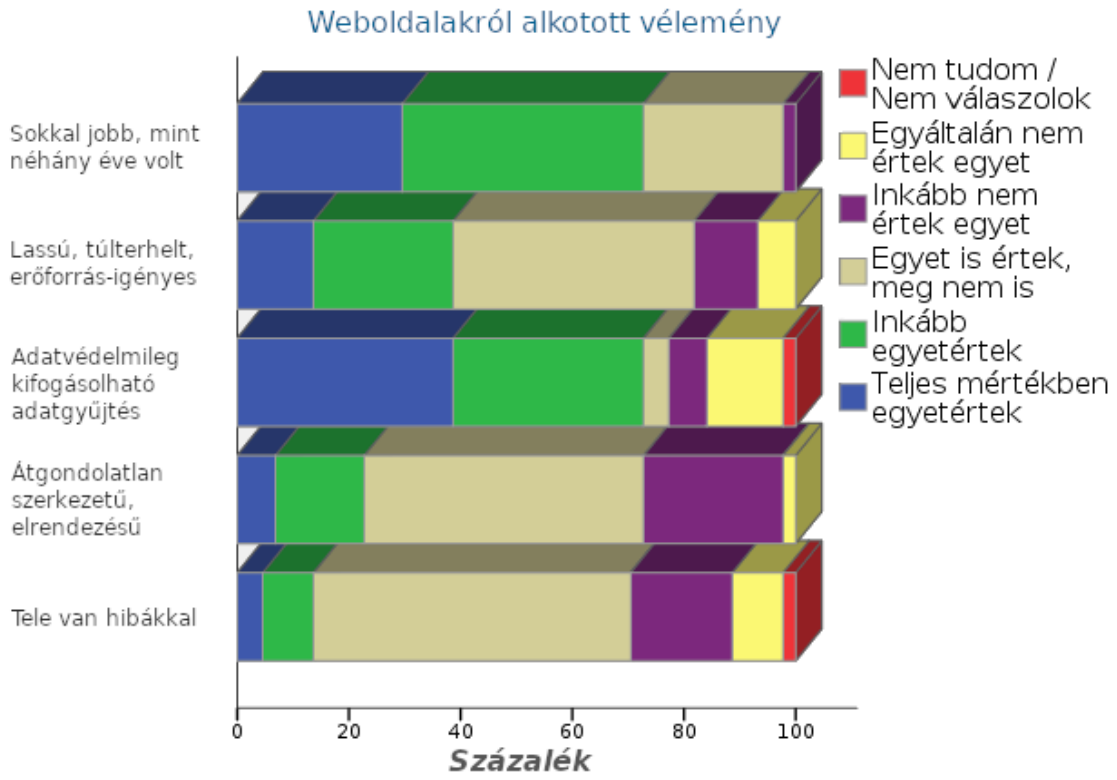
Úgy tűnik tehát, hogy a célcsoport *elfogadni látszik* azt a nézetet, hogy *komplexebb feladatok elvégzése pusztán érintőképernyős eszközökkel nem kivitelezhető hatékonyan.* Érdekes, hogy a *modern kinézettel kapcsolatos elvárás* is korlátozottan jelenik meg az elvárások között.




*11. ábra: A felsorolt szempontok fontossága a megkérdezettek számára*
*Saját grafika*

### 5.2.3. Weboldalak, üzleti szoftverek helyzete

A résztvevők jellemzően nem osztották az eléjük tárt, weboldalakkal kapcsolatos általános aggályokat. 1-től 5-ig terjedő *Likert* skálán (*1: egyáltalán nem ért egyet, 5: teljes mértékben egyetért; m=átlag, s=standard deviáció*) osztályozva *éppenhogy inkább nem értenek egyet* azzal, hogy a weboldalak *tele vannak hibákkal* (m=2.81, s=0.91), *neutrálisak* az *átgondolatlan szerkezet* iránt (m=3.00, s=0.89), és *megosztottak*, de *inkább egyetértenek* azzal, hogy *túlzottan erőforrás-igényesek* a weboldalak (m=3.27, s=1.06). Két kérdésben konkluzív a véleményük: a *mai* weboldalak minősége jellemzően *sokkal jobb, mint a néhány évvel ezelőttieké* (m=4.00, s=0.81); ugyanakkor *a "kémkedés", adatgyűjtés, GDPR kérdésköre* egyre égetőbb problémát jelent számukra (m=3.79; s=1.41).




*12. ábra: A megkérdezettek weboldalakról alkotott általános véleménye*

*Saját grafika*

Az üzleti-ügyviteli szoftvertermékek kapcsán megítélésük során sorban enyhén javuló tendenciával átlagosan semlegesnek tekintették a *hibákkal* (m=2.97, s=0.90), a *megjelenéssel* (m=2.97, s=1.17), a *felhasználóbarátsággal* (m=2.90, s=0.89), a *fejlesztők felhasználói elégedettség mellőzésével* (m=2.87, s=1.13), illetve a *fejlesztők felhasználói visszajelzések mellőzésével* (m=2,82, s=1.14) kapcsolatos felvetéseim. Viszonylagos elutasítás (m=2.67, s=1.13) kísérte a *felhasználói felület folyamatos áttervezése miatt követhetetlen a használatuk* (ez agilis módon, CI/CD mentén fejlesztett, többfelhasználós megoldásoknál valós veszély), miközben *inkább elfogadott* az az álláspont, amelynek értelmében *érdemes örülni az IT-támogatott ügyviteli integráció jelenleg létező fokának is* (m=3.51, s=1.01).

### 5.2.4. Háttérfolyamatok értékelése

A *digitalizáció* jelenségét – számomra szintén meglepő módon – a kutatásba bevontak *kifejezetten pozitívan* értékelik.

Az utolsó kérdéskör a válaszolók egyes, a technológiával álláspontom szerint szorosan összefüggő, tágabb értelemben vett folyamatról alkotott álláspontját mérte fel. Az EU-s szinten támogatott háztartási *okos mérés* kérdését alig többen tekintették *inkább* az energiamegtakarítás hasznos eszközének (40%), mint a fogyasztók *magánszférájának megsértésének* (37%), míg 23% nem válaszolt. Valamivel többen vélték úgy (32%), hogy az adataik *itthon tárolt szerveren nagyobb biztonságban vannak*, mint azok, akik a

*külföldi szervert* preferálták (22%); itt viszont igen magas, 46% volt azok aránya, akik nem válaszoltak a felvetésre.

Némileg ellentmondásosnak tűnik, hogy bár az ügyviteli megoldásoknál *kiemelt fontosságú* volt a megkérdezettek számára, hogy *felhő alapú, tehát üzemeltetés-mentes* legyen az adott termék, mégis a megkérdezettek majdnem fele (49%) szerint *a felhő alapú adattárolás számos kockázatot rejt magában*, míg 39% szerint teljesen biztonságos. (Itt meglepően alacsony, 12% volt csupán a nem válaszolók aránya.) Az előzetes várakozásom szerint ez arra volna visszavezethető, hogy ugyan *elismerik, hogy a felhő több szempontból kockázatos, de a munkaerő-megtakarítás adta előnyt nem tudják mellőzni.* A két változó közötti összefüggés Pearson-mutatója azonban r = -0.41, p=0.007 szignifikancia-szint mellett; a mérsékelt negatív összefüggés arra enged következtetni, hogy *minél komolyabbra értékeli a felhős adattárolás veszélyeit* az érintett, *annál kevésbé akar vele munkaerő-költséget megtakarítani*.

Érdekes az is, hogy – a várakozásommal ellentétben – a válaszolók pont fele (50%) úgy véli, hogy a NAV adatszolgáltatás *a gazdaság fehérítését szolgálja* elsődlegesen, és csupán 23% véli úgy, hogy ez *aggályos lenne az üzletmenete szempontjából például bennfentes információszerzés miatt*. (Igaz, itt is 27% volt az állást nem foglalók aránya.)

Számomra a legmeglepőbb azonban az, hogy – mondható: a fentiek ellenére – 53% úgy véli, hogy inkább a *legfrissebb technológiára* áldoz *bármi áron*, és csak 35% mondta azt, hogy *inkább a helyi gazdaságot támogatná* megrendelésével. Itt is alacsonynak mondható (12%) a *tartózkodók* aránya.

### 5.2.5. A kérdőív következtetései

Az első hipotézist, amely értelmében *célcsoport* ügyviteli integrációs fokának *alacsony* szintjében a *belső kockázatok* szerepe *elhanyagolható*, sikerült különösebb vizsgálódás nélkül, *egyértelműen* igazolni.

A feltárt szentimentumok és attitűdök alapján *egyértelműen* megállapítható, hogy *a kutatásban résztvevők csoportja* a kérdések alapján *nem mutatott olyan averziót (elzárkózást), amely a korábbi vagy jelen idejű, a szoftverek minőségére vonatkozó negatív tapasztalatára volna visszavezethető.* Ez a résztvevők a vállalati tartomány viszonylag széles skálájához tartozó hátterét tekintve mindenképpen mérvadó.

Ez egyben azt is jelenti, hogy *lehetséges* itthon, technológiailag olyan szoftverterméket fejleszteni, amely vonzó *lehet* valamely, a célcsoporttal megegyező döntéshozó és/vagy felhasználó számára.

Az természetesen lehetséges, hogy a (például *életkor, vagy lakhely* mentén értelmezett) *reprezentativitás* kapcsán egyéb, nem feltárt faktorok játszanak szerepet az *alulreprezentált* csoportok vizsgált jellemzőjének alacsony szintjében; az azonban *nem vélelmezhető*, hogy ezek a jelenleg elterjedt termékek átlagos minőségére vezethetőek vissza. Az EU által támogatott, áttekintett kutatások alapján is kialakított álláspontom szerint elsősorban társadalmi vagy kulturális, illetve infrastrukturális nehézségek,

különbségek okozhatják. Mindenesetre ezek olyan szempontok, amelyeket valamely szoftver *fejlesztői oldaláról* kiküszöbölni *nem lehet*, ezért belső kockázatnak nem tekinthetőek.

A második hipotézisnél, vagyis hogy a *célcsoport* IT-támogatott ügyviteli integrációs fokának *alacsony* szintjében a *külső kockázatok* szerepe *elhanyagolható*, kevésbé egyértelmű a helyzet.

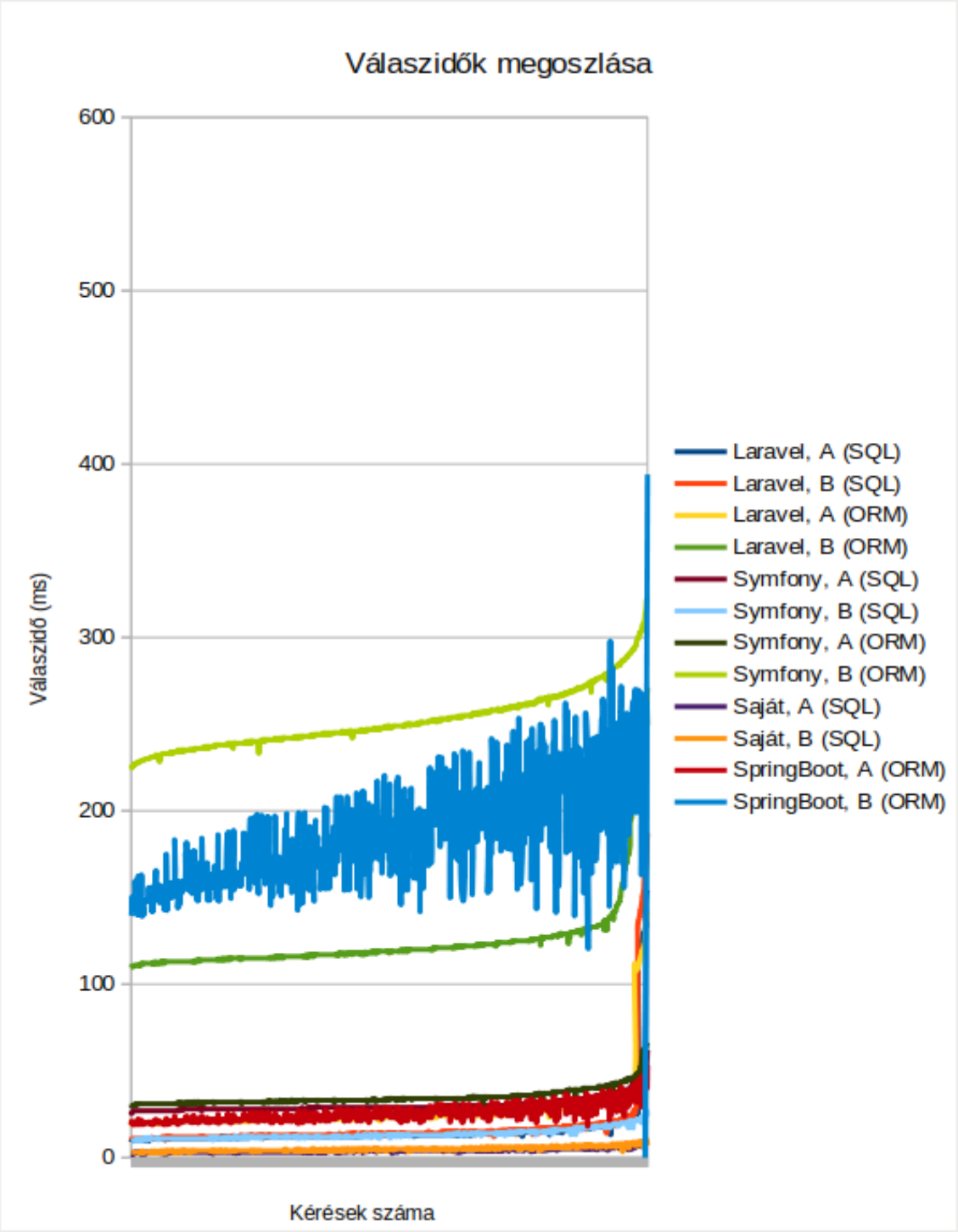
Egyrészről természetesen *ki lehet azt mondani*, hogy *a második hipotézis egyben megcáfolásra került*, ugyanis a kockázatok két halmaza komplementer, vagyis együttesen alkotják a kockázatok *teljes halmazát*; ebből kifolyólag pedig *a kettő közül legalább egy hipotézis megcáfolásra kell, hogy kerüljön.* Másrészt mindemellett nem egyértelmű, hogy a külső kockázatok között *milyen arányban* oszlanak meg a dolgozat keretein belül vizsgált, illetve nem vizsgált kockázatok.

A kérdőíves kutatás végkövetkeztetése tehát az, hogy a piacon lévő termékek minősége *nem* domináns az elterjedésük viszonylagosan alacsony szintjében. Ez egyben azt is jelenti, hogy egy új termékkel kapcsolatban technológiailag *nem igen magas a belépési küszöb*, ennek a megugrása azonban *nem is eredményez szükségszerűen piacot.*

## 5.3. A vizsgált technológiák összehasonlítása

A megvizsgált platformok teljesítménye között számottevő, többszörös, de nem nagyságrendi eltérés tapasztalható. Általánosságban véve elmondható, hogy a minél kevesebb indirekciós réteg, a minél *natívabb*, gépi kódban megvalósított halmaz- és gyűjteménytranszformációk, attribútum-átalakítások, valamint a *tükrözés* helyett a nyelv és platform által alapvetően biztosított adattöltési mechanizmusok kedveznek inkább a teljesítmény-maximalizálásnak.

A mérési eredmények összegzése a *13. ábrán* látható. Ezek alapján a legkisebb válaszideje a *saját megvalósításomnak* van, átlagosan *4 ezredmásodperc*. Ez nem véletlen: ennek fejlesztésénél *kifejezetten a teljesítmény, a sebesség* volt a fókuszban, *természetesen a hasonló sebességű megoldások többségével szemben a megfelelően felépített struktúrájú kódolás támogatása mellett.* Cserébe azonban nyilván egyes, összetettebb rendszerekre jellemző megközelítéseket – mint például az *eseményvezérelt kéréskiszolgálás*, vagy az *Enterprise Service Bus* architektúra – nem, vagy csak aránytalan többletmunka árán lehetne benne megvalósítani.
Ez a különbség azonban szinte elhanyagolható a *Laravel – ORM* nélküli, *lekérdezés-generátoros* – megközelítésének a *17 ezredmásodperces* átlagos válaszidejéhez képest. Jóval számottevőbb azonban az olyan megoldások, mint az *ORM* hatása. Ez már az egyszerű, közvetlen rekordkészlet lekérdezésén alapuló teszt-esetnél is átlagosan 26 ezredmásodpercre növelte a (ugyanabban a keretrendszerben, ugyanannak az adatnak ugyanabból az adatbázisból történő lekérdezésének) válaszidejét. Igazán szembetűnő azonban az *aggregát* (készlet-információt termékenként több sorból összesítő) modell: itt az átlagos válaszidő *124 ezredmásodperc*.




*13. ábra: A vizsgált technológiák válaszidejeinek alakulása a kérések számával*

*Saját grafika*

Hogy érdemes-e akkor *saját keretrendszert* fejleszteni? Ez *attól függ*. Sokan azért állnak neki saját megoldást használni, mert el akarják kerülni a megismerkedést valamely elterjedt alternatívával (jellemzően azok meredekebb tanulási görbéje miatt). *Az ilyen esetekben szinte borítékolható a katasztrófa.* Ha bármilyen esetben *érdemes* ilyesmibe vágni, az leginkább a már meglévő megoldások *viszonylag alapos* megismerését *követően* indokolható; minden más esetben bizonyosan inkább növelni fogjuk a kockázatot, mintsem csökkenteni. Arról nem beszélve, hogy milyen *biztonsági*

*kockázatot* jelent tapasztalat nélkül egy saját megoldásnak nekilátni. Emellett a meglévő megoldások többsége *teljesen ingyenes és kereskedelmi célra is szinte korlátozás nélkül felhasználható licencű.*

*V*agyis összességében valószínűleg *sem biztonságosabb, sem gazdaságosabb* megoldást nem sikerül majd előállítani – tehát ha *bármi* okból érdemes, az az ok aligha fog a felsoroltak közé tartozni.

A benchmarking érdekessége számomra, hogy a Java, a SpringBoot keretrendszerrel, Hibernate ORM alapon sebességben *mindkét esetben a PHP Symfony-Doctrine* szintjét *némiképp* meghaladóan teljesített. Ennek értékelhetősége kettős: sokaknak – akik vélhetőleg *az Interneten fellelhető, tudományosan megalapozatlan, aluldokumentált és reprodukálhatatlan benchmark-kísérleteknek* hittek korábban – jó eséllyel csalódást jelenthet; az én személyes tapasztalatom a Java-val üzemeltetési, felhasználási szempontból viszont *rendre az eltúlzott erőforrás-használatra* vezetett – ebből kiindulva számomra teljesen meglepő, hogy a Java EE 8 teljesítménye Hibernate-el abszolút összemérhető egy alsó-közepes teljesítményű kiszolgálói környezetben is egy olyan PHP technológiával, amely *le se tagadhatná* a Hibernate-inspirációját. *Ez egyben azt is jelenti, hogy amennyiben technológiai vagy domain-tervezési okokból ragaszkodunk az ORM alapú perzisztencia használatához, a Java megoldás abszolút versenyképesnek tekinthető a kevésbé tőkeerős üzleti szegmensben is.* Felettébb érdekes ugyanakkor, hogy a Java-alapú platform válaszideje *óriási*, a PHP-t *jóval meghaladó mértékben* szórt, és ez a szórás a kérések számával (*vagyis az erőforrás-kihasználtsággal*) arányban nőtt. Emiatt szükséges lehet megvizsgálni, hogy valamely adott szolgáltatás-hardver kombináción mi az a legnagyobb terhelés, amely felett a kérések egy meghatározott része meghaladja az elfogadható válaszidőt. A PHP-nél érdekes módon ez a jelenség nem ábrázolódott; a válaszidő szórásának gyakorisága és mértéke is jóval alacsonyabb volt. *Ez véleményem szerint a garbage collector működésével hozható összefüggésbe.*

Ha saját keretrendszert írni nem is, valamit *biztosan érdemes megtenni:* ez pedig az a különböző gyorstalpalók, YouTube tutorial-ok és az *evangélisták* által összegzett, gyakran félreértelmezett "best practice" megoldások helyett *megvizsgálni* a problémát, amelyet meg szeretnénk oldani, *majd ezt követően* mérlegelni, hogy egy adott, ilyen módon megismert megközelítés jelent-e akkora előnyt, hogy a *kétségtelenül meglévő* hátrányát ellensúlyozza – *hátránya ugyanis mindennek van*.

Álláspontom szerint a SOLID elvek, különösen a *szétválasztott felelősség elve* esetén ez nem kérdés – hiszen a hátrány itt nagyjából némi extra fejlesztési teendőre és némi előrelátásra korlátozódik.

Egy olyan esetben viszont, mint az *ORM* használata, ugyanez *messze nem ilyen egyértelmű*. Elsőre ugyan vonzónak tűnik, hogy *a rendszer automatikusan* konvertálgatja az entitásokat objektummá és vissza, és *még lekérdezést sem kell tudni írni* – de érdemes észben tartani, hogy itt *néhány fejlesztői munkaóra megspórolása*

ellenében esetleg *évekig tartó üzemeltetési extra kiadást* biztosíthat egy-egy nem megfelelően választott megközelítés.

Az eddigi, ebből a szempontból legextrémebb helyzet az egyik korábbi munkahelyemen állt elő: történt, hogy Symfony rendszerben lefejlesztettek egy projektet, ahogy az *a best practice* javaslatokban le van írva, mikroszolgáltatás-architektúrára, több rétegben. Működött is a rendszer, ahogy kell, csak *viszonylag lassú* volt: annak ellenére, hogy DevOps szemléletben, az Amazon felhőjén működött, terhelés-elosztással meg mindennel, a fő folyamat 1-1.5 másodpercet vett igénybe minden esetben. Természetesen tele volt rakva mindenféle indokolatlan komplexitással; *sokkal inkább tűnt egy szenior rendszertervező vizsgafeladatának*, mint egy gyakorlati használatra tervezett rendszernek. De nem ez volt az igazán meglepő, hanem az, hogy hosszas töprengés után az lett a konklúzió, hogy *a PHP lassú, újra kell írni az egészet Node.js-ben.*. Én a magam részéről együttérzően bólogattam, és közben valami olyasmi járhatott a fejemben, hogy *valóban, például zuhanyzáshoz minden bizonnyal a porszívó az egyik legrosszabb csavarhúzó*.

Gyakorlatilag tehát az egyik legfőbb okot egy technológia nem kellően alapos mérlegelést, megismerést követő kiválasztása mellett – az időhiányt – érdemes *azonnal* kizárni: kellő mérlegelés, elemzés, mérés nélkül váltani valamelyik *hype* fázisú technológiára *biztosan* előre nem látható időveszteségeket, strukturális vagy teljesítményproblémákat fog okozni, *és jó eséllyel akkor*, amikor már az *újabb, egyébként szükségesnek ítélt átállás* egy másik technológiára gazdaságilag nehezen indokolható.

Mindezek értelmében az utolsó hipotézist – miszerint a rendelkezésre álló *legújabb* fejlesztési, tervezési módszertanok és technológiák, valamint az ezek *fejlesztői* által javasolt *best practice-ek együttesen garantálják* egy magas *fajlagos teljesítményű*, webes felületű, üzleti-ügyviteli célú alkalmazás rövid idő alatti, *alacsony kockázatú* fejlesztését – *meg kell cáfolnom*: a technológiák közötti választás *igen magas szintű körültekintést* igényel, és ezt az azokhoz tartozó legelterjedtebb gyakorlatok (amelyek mögött sokszor változatos, *nem szükségszerűen szakmai motivációk* húzódnak meg) a legtöbbször egyáltalán nem ellensúlyozzák.

# 6. NYITOTT KÉRDÉSEK

A kutatás során számos olyan helyzetre, szempontra derült számomra fény, amely további vizsgálatra lehet érdemes.

Először is érdemes lenne a megkérdezettek véleményét *kontextusba helyezve* összevetni külföldről származó hasonló adatokkal. Több esetben például meglepődtem a válaszokon, mert azokat a korábban megismert irodalmi adatok, illetve a marketingen keresztül kialakított *köztudat* nem

További vizsgálatot igényelne például, hogy fény derüljön arra, hogy *milyen arányban* vannak a 30 éves kornál fiatalabb *döntéshozók, vezetők* itthon, és külföldön, annak érdekében, hogy meg lehessen vizsgálni, hogy a *mobiltelefon, mint elsődleges számítógép-helyettesítő hype-korszaka ténylegesen* lejárt, vagy egyéb, például életkor alapú sajátosságokra visszavezethető okokból jelent meg ez az eredmény. Személy szerint az *előbbit* abszolút nem tartanám meglepőnek.

Érdeklődésre tarthat okot megvizsgálni azt is, hogy amikor a válaszadók többsége mind az *on-site*, mind a felhős működést *inkább fontosnak* vagy *nagyon fontosnak* tartja, akkor ezt egy terméken belül, *offline* is működtethető, "szigetüzemű" rendszerrel, vagy külön termékekre vonatkoztatva képzelik vajon el, esetlegesen a választás lehetőségét (én magam ez utóbbit *eleve támogatom*, és szándékomban áll ennek a fenntarthatósági-licencelési kérdéseit megvizsgálni, már csupáncsak a SPSS Linux verziójának a licenckezelési nehézségei kapcsán, technológiai oldalról is).

Ennél szélesebb körű és talán nagyobb közérdeklődésre számot tartó kérdés véleményem szerint például, hogy mivel a válaszadók többsége számára a *hazai gazdaság fenntartása másodlagos*, mivel lehetne a gazdasági szereplők versenyképességét és vonzerejét javítani a *belföldi, régiós* gazdaság szereplői előtt. Az már most látszik, hogy ez a *kölcsönösség* alapján működhet kizárólag, és ezért ismeretterjesztés, tájékoztatás nélkül aligha képzelhető el. *Lehetséges azonban, hogy az ezen felvetés által érintett szempontok meghaladják az informatika, az információ-technológia, és esetlegesen a gazdaság kereteit is.*

A technológiai összehasonlításnál sem *értem el a képzeletbeli plafont* a teljesítmény-vizsgálatban. Lenne létjogosultsága megvizsgálni az *írási* teljesítményt; de például komplexebb teszt-esetek alkalmazására, illetve több, eltérő típusú adatbázismotor összehasonlítására is sort lehetne keríteni. Emellett tanulságos lenne a *front end* technológiák teljesítményének vizsgálatát is napirendre tűzni – itt azonban a különböző prezentációs technológiák összehasonlíthatósági célú standardizálása, illetve a különböző böngészős komponensekkel való interakciók teszteléshez nélkülözhetetlen automatizálása, vagy ezzel kapcsolatban megbízhatósági célból a böngészőről leválasztott időzítés is összetett kérdéseket rejt magában.

# 7. ÖSSZEGZÉS

Annak ellenére, hogy nemzetközi viszonylatban a vezérigazgatók megkérdezésével végzett kutatások, és a napi szakmai hírek is arra utalnak, hogy *számos dolog nincs rendben* a technológiai szektorban, és kevés, illetve csökkenő mennyiségű ok van az optimizmusra, a hazai helyzet ezt nem tükrözi vissza maradéktalanul.

A megkérdezettek technológiai, internetes szolgáltatásokhoz, illetve üzleti szoftverekhez kapcsolódó szubjektív értékítélete is alapvetően egybevág a hazai, optimistának tekinthető szentimentummal. Érdekesség ugyanakkor, hogy a felhőszolgáltatások kockázataihoz kapcsolódó félelmek, illetve az azokkal elérni kívánt (elsősorban munkaerő-) költségmegtakarítás egymással fordított összefüggésben van; elképzelhető tehát, hogy ezen kockázatok részletesebb megismerése ezen asszociáció szorosságát erősítené.

Ma Magyarországon internetes üzleti-ügyviteli szoftvert fejleszteni a siker reményével *nem lehetetlen*. Azt is láthattuk, hogy a megkérdezetteknek *nem lehetetlen megfelelni*. Alapvetően kevésbé szigorú elvárásokat támasztottak a szoftvertermékekkel és weboldalakkal szemben, mint amelyet én fejlesztőként elvárhatónak tekintek.

Az ugyanakkor, hogy az érintettek minden optimizmusa ellenére is *az IT-támogatott ügyviteli integráció elterjedése alacsony szintű és lassan növekvő*, arra enged következtetni, hogy a háttérben meghúzódnak olyan, a *konjunkturálistól különböző* kockázatok, amelyek befolyással lehetnek egy integrált üzleti-ügyviteli megoldás piacszerzési lehetőségeire. Ezek jelen dolgozat keretében feltárásra nem kerültek; az irodalmi adatok alapján valószínűsíthető, hogy a *város-vidék ellentét*, a *digitális készségek hiánya* éppúgy megjelenhet ezek között, mint a szürke- vagy éppen feketegazdaság mértéke. Kérdés ugyanakkor, hogy ezen kérdések kapcsán egy szoftverpiaci szereplőnek milyen feladatai vagy egyáltalán lehetőségei vannak, lehetnek.

A rendelkezésre álló technológiák *korábban soha nem látott lehetőségeket* biztosítanak, azonban erre is igaz, hogy, *mint a devizapiaci kereskedés, a technológia-választás is magas kockázatú tevékenység*. Egy *megfelelően kiválasztott* technológia, annak *megismerését* követően, kiválóan tudja támogatni *bármely* rendszer fejlesztését, amelyhez az alkalmas.

*Pusztán a népszerűség ugyanakkor semmilyen körülmények között nem tesz alkalmassá bármilyen technológiát vagy eszközt egyetlen feladatra sem.*

Hasonlóan nagy probléma az *internetes óriásvállalatoknál való fejlesztésre, vagy épp elterjedtségre* való hivatkozás. Az, hogy egy technológiát egy nagy, tőkeerős piaci szereplő használ, az éppúgy jelentheti annak nagyszerűségét, mint azt, hogy *a bevezetésre, átállásra már felhasznált erőforrásokat nem lehet arcvesztés nélkül leírni.*

Érdemes csínján bánni az úgynevezett *best practice* jelleggel tálalt megközelítésekkel is; ezek jelentős része ugyanis *igen egyoldalúan* közelíti meg a témát. Fontos észrevenni, hogy a feltörekvő *webes* platformok (legyen szó PHP-ról vagy éppen Ruby on Rails-ről) a leggyakrabban a Java nagyvállalati (enterprise) változatánál kikísérletezett módszereket és mintákat igyekszik *másolni*, eközben elfelejtve azt, hogy *azoknak elterjedése – amellett, hogy legalább 10-15 évvel ezelőttre tehető – általánosságban véve legalább annyira volt köszönhető újszerűségüknek és piaci elsőségüknek*, mint hatékonyságuknak vagy éppen valamely feladatra való alkalmasságuknak.

A legtipikusabb példa az objektum-reláció-leképezés (*ORM*) kérdése. Megfelelő *gazdag (rich) domain* esetén a *javaslat* az, hogy forrásnyelvi kódból hozzuk létre az *aggregát objektum-csoportot* – célszerű lehet elmerengeni ugyanakkor azon, hogy ezt egy valós időben értelmezett, nem natív kódra fordított nyelvben vajon mennyivel lehetséges hatékonyabban megvalósítani, mint egy bizonyosan több évtizede kifejezetten erre a célból fejlesztett, *natív (gépi) kódú* relációsadatbázis-motor.

Abba is hasznos belegondolni, hogy például ReST vagy SOAP technológia esetében *alapvetően külön objektumok* közlekednek a HTTP protokollon keresztül, miközben sokszor az adatbázisban való tárolásuk és a megjelenítésük is hasonló, *táblázatos* formában történik. Ekkor valójában szinte mindig az *értelmezett nyelvű* kód fogja azt valamely HTTP-kompatibilis formába alakítani, *pusztán azért*, hogy a célhoz érkezését követően *rögtön* visszaalakuljon akár az eredeti formátumára – szintén az *értelmező* extra erőforrás-igénye mellett.

Ezek mind olyan problémák, amelyek valamely vélelmezett egyszerűség, vagy néhány fejlesztői munkaóra megtakarítása címén kiválóan *alkalmasak* arra, hogy *évekig tartó ismétlődő* üzemeltetési költséget generáljanak az érintett megrendelőnek vagy üzemeltetőnek, sok esetben jóval az érintett fejlesztő vagy tervező távozását követően is.

# 8. KIFEJEZÉSEK JEGYZÉKE

*Itt gyűjtöttem össze azokat a kifejezéseket, amelyek a magyar nyelvbe, vagy a köztudatba még nem kerültek bele, esetleg kontextusfüggő, egyedi és/vagy a megszokottól eltérő értelmezésben szerepelnek a szövegben.*

***abuse compliance***: valamely online szolgáltatás esetében annak visszaélésszerű, jellemzően harmadik fél nyugalma, biztonsága, vagyona vagy egyéb joga ellen irányuló felhasználását, az ilyen használatok feltárását segítő jogi intézkedéseknek való megfelelés, ennek érdekében az azokban kijelölt szervekkel való együttműködés.

***agilis módszertan [szoftverfejlesztés]***: egy, számos neves szoftverfejlesztő /-mérnök által 2001-ben életre hívott kezdeményezés kiáltványában ("*Agile Manifesto*") összefoglalt célkitűzések és módszerek összessége. A legfőbb elve a korai és folyamatos szoftvertermék-szállításon keresztül a megrendelő elégedettségének biztosítása (BECK, K. ET AL., 2001). Emellett olyan, korábban hátrányba szorult célkitűzések mentén működik, mint a *lelkesedés a követelmények változásaival szemben, a kiadások legalább néhány havonta történő biztosítása,* az egyszerűségre (az *el nem végzett fölösleges munka mennyiségének minimalizálása)* való törekvés, vagy például – ami kölcsönös üzleti szempontból a legfontosabb – *a fejlesztők és az üzlet folyamatos közös munkája, egyeztetése – ez* utóbbi bármely, előzetes megrendelésre történő, fejlesztés esetén a megrendelő szükségszerű bevonásával jár.

***backlog***: az *agilis módszertanban* (és máshol) az el nem végzett, viszont továbbra is szükségesnek vélt feladatok, valamint a feltáratlan vagy megoldatlan problémák (együttesen: *technikai adósság*) jegyzéke. A legegyszerűbb megvalósítás lehet egy szöveges állomány vagy egy táblázat; a gyakorlatban a legtöbbször úgynevezett *feladatkövető (issue tracking) rendszert* alkalmaznak, amely a problémák rögzítése és kategorizálása mellett a kapcsolódó adatok (pl. állományok, képek) csatolását, valamint kommunikációs (hozzászólási, esetleg csevegési) lehetőséget is biztosít.

***bérkényszer [humánerőforrás-menedzsment]***: valamely foglalkoztató kényszere arra, hogy – ütemezés betartása, előírt létszámkeret biztosítása, vagy éppen a többi beosztott munkaköri leírásán történő túlterjeszkedés elkerülése érdekében – *emelt, a piacit meghaladó, vagy saját szokásos vagy tervezett bérsávján felüli* – juttatási csomag mellett is *új munkaerőt* alkalmazzon, vagy a meglévő megtartása érdekében az ő juttatásukat növelje. Ez elsősorban munkaerő-hiányos makro- (földrajzi) vagy mikrokörnyezetben (piaci szegmens) fordulhat elő. Két fő járulékos kockázata, hogy új munkaerő bevonása esetén *bérfeszültséghez* vezethet a többi, azonos vagy hasonló munkakörben foglalkoztatott dolgozó között (amennyiben az ő bérük nem kerül egyidejűleg *kiigazításra*); illetve jellemzően nem egyeztethető össze egy esetleges egyidejű *költségkényszerrel* (ami más, a problémával esetleg kevésbé érintett területeken történő leépítésekhez, át-/kiszervezésekhez vezethet).

***buzzword***: *divatos, trendi* kifejezés, amelyről mindenki, aki éppen *számít*, beszél. A technológiában az éppen népszerűségnek örvendő jelenségek, koncepciók, fogalmak megnevezése is.

***csapatkockázat [szervezeti magatartás]***: azon kockázatok összessége, amelyeket egy adott üzleti (*a dolgozatban: fejlesztési, integrációs*) (rész)folyamatban szerepet és/vagy felelősséget vállaló csoport(ok) tagjai valamely, a folyamatra ható tevékenységük során, vagy annak hiánya által egymás, illetve a célkitűzések irányában jelentenek. Ide tartoznak egy csapat együttműködési készségével, teljesítményével, szakmai és személyi kompetenciáival kapcsolatos kockázat, de ide sorolható a humánerőforrás-menedzsment metodikájából fakadó esetleges *kontraszelekció*, munkaerő-hiány, vagy egy adott pozíció egyéb okból fennálló betölt(het)etlensége jelentette bizonytalanság is.

***deklaratív programozás***: azon koncepció, amely keretein belül a *programozás* úgy zajlik, hogy azt *"mondjuk meg"* a gépnek, a keretrendszernek, a környezetnek, hogy *mit* szeretnénk [számolni], azt nem, hogy *hogyan* (TORGERSSON, O., 1996). Ez például egy jól definiált kontextusban (mint mondjuk az SQL) tud működni *az adott kontextus előzetesen megállapodott keretein belül*. Minden további feladat azonban az *imperatív tartományba* történő átlépést igényli (SQL esetén ilyen például a PL/SQL, vagy a tárolt eljárások alkalmazása). Torgersson a fenti definíció ellenére úgy ítéli meg, hogy mind a *gyenge*, mind az *erős* értelmezésben a *logika* átadása mindenképpen a része a deklarativitásnak; ez álláspontom szerint ellentmondásos, ezt később, a már *definíciós programozásként* nevesített fejezetben fel is oldja azzal, hogy *sajnálatos módon* a programozónak mindenképpen *tisztában kell lennie* az *irányítási, vezérlési problémakörrel*.
Mindenesetre a formális logika szabályainak értelmében a *folyamat leírásának birtoklása* nélkül nem képzelhető el semmilyen ilyesféle deklaratív metodológia működése; eltérés abban van, hogy ez a *folyamatleírás* előzetesen kerül a környezetbe beépítésre, vagy a deklaráció során átadásra annak részére. Én a második esetet jelen dolgozat keretei között nem sorolom a *deklaratív programozás* tárgykörébe.

***DI konténer***: olyan fejlesztési minta, amely a *singleton* mintához hasonlóan lehetővé teszi az alkalmazás belső szolgáltatás-osztályai egyes példányainak a tárolását és visszaigénylését, illetve – hiányzó példány esetén – létrehozását és konfigurálását. Előnye azzal szemben, hogy mivel nem *statikus* megvalósítású, ezáltal a példányok és a konténer is szabadon felcserélhetőek például *próbapéldányokkal*, amelyek a unit tesztelést segítik.

***diversity and inclusion [szervezeti magatartás]***: törekvés, melynek lényege, hogy a csapattagok között a különböző védett kisebbségek (hölgyek, külföldi származásúak, megváltozott munkaképességűek, stb.) számaránya *számottevően* növekedjen. A törekvés létjogosultsága megkérdőjelezhetetlen, módszertanilag azonban az érintettek között is ellentmondásos az alkalmazott megközelítések megítélése.

***due diligence***: elvárható gondosság / előrelátás. Annak a mértéke, hogy egy adott üzleti döntési folyamat felelősei a pozíciójukkal járó gondosság és/vagy helyzetfelismerés

követelményeinek eleget téve jártak-e el az adott döntés meghozatala során. Alapvetően megfeleltethető a *polgári törvénykönyvről szóló 2013. évi V. törvény (Ptk.) 1:4. § (1)* szerinti elvárhatósági kritériumnak: *"Ha e törvény eltérő követelményt nem támaszt, a polgári jogi viszonyokban úgy kell eljárni, ahogy az az adott helyzetben általában elvárható."*

***fajlagos teljesítmény [szoftverteljesítmény-mérés]***: az *eltérő technológiával és/vagy platformon*, de *azonos funkcionalitással* megvalósított, *azonos infrastruktúrán* üzemeltetett alkalmazások valamely funkciójuk vagy ezek csoportja kiszolgálása során nyújtott relatív teljesítmény. Fontos szempont, hogy a *funkcionalitással* ellentétben (amely legtöbbször a fejlesztés bemeneti követelménye, ekként *adott*) az *infrastruktúra* kiválasztása – külső követelmény híján – *objektíven* történjen, azaz ne valamely konkrét mérni kívánt *platform* által előnyben részesített infrastrukturális környezet kerüljön kiválasztásra, hiszen az torzítaná a mérés eredményét.

***fenntarthatósági kockázat***: a strukturális fenntarthatóság adott időkereten belüli megszűnésének valószínűsége; jelen definíció szerint a strukturális fenntarthatóság valószínűségi komplemense

***folyamatos integráció (continuous integration)***: az agilis módszertan egyik eleme, a *DevOps* szemlélet alapvető előfeltétele. Lényege, hogy a *szolgáltatásként hozzáférhető* projekt forráskódja *folyamatosan* frissül, tehát nem előre, fixen meghatározott *kiadási ütemterv* van, hanem az elkészült funkciók rövid időn belül, a hibajavítások pedig azonnal *beépítésre (merge) kerülnek* a ténylegesen használt "éles" forráskódba. Csaknem elengedhetetlen feltétele, hogy *unit testing* keretében a legtöbb egység le legyen "fedve" automatikusan futtatható tesztekkel, ezáltal a kód szintaxishibái mellett a szemantikus, illetve algoritmikus hibák is passzívan, beavatkozás nélkül felismerhetőek legyenek (*ez a leggyakrabban a korábban már javított, illetve az előre vélelmezett hibákra szokott működni*). A folyamatos integráció megvalósítása valamely technikai eszközzel (CI eszköz) történik, amely a folyamatot konfigurálhatóvá teszi. Egy úgynevezett *beépítési kérést (merge request)* követően a CI eszköz lefuttatja a teszteket, és ha nem talál hibát, beépíti a kódba a módosításokat, majd a kombinált kódot kihelyezi a végleges helyén (*deployment*) olyan módon, hogy az esetleges kiesés minél alacsonyabb legyen. Hátránya például, hogy főszabályként a *leggyorsabb* hibajavítás a tesztek futtatásának és a kihelyezés (*deployment*) együttes időtartama (*jellemzően ez percekben mérhető; komplex projekteknél, illetve erőforrás-korlát esetén a 30-60 perc abszolút nem elrugaszkodott*). Ez nem jelent *szolgáltatás-kiesést*: a kihelyezés ideje alatt a régi verzió működik; az esetleges kiesés *tényleges ideje* optimális esetben a másodperc törtrésze lehet.

***függőség-megfordítás***: a SOLID irányelvek "D" betűje, Dependency Inversion. Lényege, hogy a függőségeknek a konkréttól az absztrakt irányába kell mutatnia, vagyis az absztraktabb osztály nem függhet az egzaktabb megvalósítástól. Ennek gyakorlati nehézségei miatt ezt korábban a legtöbbször a *szolgáltatásjegyzék* mintával oldották meg; mára inkább a *DI konténer* megvalósítás az elterjedt.

***hype***: irracionális, eltúlzott, sokszor indokolatlan lelkesedés, rajongás valamilyen közösségi, spirituális vagy éppen természettudományos jelenség iránt. A legtöbb *hype*-ot a jelenség iránti közöny (*feledés* homálya) vagy az azzal kapcsolatos kiábrándultság követi (*korrekció*).

***interface [szoftverfejlesztés]***: kettős jelentéssel bír.
Az objektum-orientált programozáson belül osztálymetódusok egy csoportjának a definíciója, amely lehetővé teszi, hogy az azt *megvalósító* osztályok az adott *interface-t* hirdető egyéb komponensekbe, osztályokba egymással csereszabatos módon behelyettesíthetőek legyenek.
Emellett a többszintű, vagy elosztott rendszerek szolgáltatás-elemei közötti kommunikációra használt protokollok, paraméterek és formátumok egy adott szolgáltatás-elemre vetített összessége.

***kockázat [minőségbiztosítás]***: a bizonytalanság hatása valamely szervezet folyamat, termék, stb. céljainak elérésére, vagyis arra, hogy a vele szemben támasztott előzetes elvárásoknak megfeleljen. Az ISTQB ennek alapján a szoftver-minőségbiztosítási ~ot projekt~ra (ez az ISO szerinti folyamat~), továbbá termék~ra osztja.

***kontraszelekció***: egy vagy több adott üzleti folyamat megvalósítására létrehozott csapat valamely tagjainak olyan kiválasztási folyamata, amely során a hivatalosan meghirdetett célkitűzés megvalósításához vélelmezhetően szükséges kompetenciák hátrányba kerülnek más, személyes, implicit, és/vagy a folyamat szempontjából alárendelt jellemzőkkel szemben.

***költségkényszer (cost pressure)***: a versenyképesség, a *strukturális fenntarthatóság* megőrzése vagy növelése céljából történő, a költségek minden egyéb szempontot megelőző leszorítása iránti, jellemzően külső vagy annak vélt hatás.

***legacy code***: réginek, idejétmúltnak, elavultnak tekintett programkód. Számos esetben valóban elavult, tervezési mintákat, irányelveket, módszertanokat mellőző architektúrákról van szó; egyre gyakoribb azonban, hogy a 2-3 évvel ezelőtt készített kódokat egyes *elitista* fejlesztők *legacy-nek* bélyegzik, hogy *új technológiákkal* dolgozhassanak, ismerkedhessenek meg; ebben látják ugyanis a *magasabb bérezés* reményét. Ennek következménye kettős: egyfelől ameddig valamely fejlesztő folyamatosan az új technológiákat keresi, addig *nem lesz középszinten sem tapasztalt egyikben sem, pláne nem senior*. (Tapasztalatom értelmében emiatt rengeteg, *kulcsszó*-szinten jól informált, ugyanakkor *mélységekben* tapasztalatlan fejlesztő jelent meg a jelen pillanatig a piacon.) Ugyanakkor a jelenlegi, munkaerő-hiányos környezetben ez azzal is jár, hogy ha egy fejlesztésben érdekelt piaci szereplő nagyjából három éven át nem korszerűsít, nem *refaktorál* valamely terméket, ledolgozandó a *technikai adósságot*, akkor egy fejlesztő munkatárs esetleges távozását követően nem feltétlenül csak az esetleges bérezés kérdése miatt nem talál majd jelentkezőt.

***mikrovezérlő [hardver-technológia]***: olyan, egy egyszerű *számítógépre* hasonló digitális technológiai megoldás, amely egy *mikroprocesszort*, a perifériák vezérléséhez

szükséges áramköröket, és a legtöbbször, elsősorban adatbiztonsági vagy költséghatékonysági szempontból *tartós (flash/EEPROM) és átmeneti (RAM) memóriát* is tartalmaz az áramkör tokozásán belül beépítve. Ma, amikor egyre több gyártó jelenik meg *perifériákkal (de nem memóriával) egybeépített (pl. ARM alapú) processzorokkal* (ezek neve SoC, System-on-Chip, azaz *áramkörbe integrált rendszer*), a mikrovezérlők fő megkülönböztetője a *beépített programtár (tartós memória)*. Jellemzően ilyen *mikrovezérlők* irányítják a legtöbb modern háztartási / fogyasztói elektronikai eszközt (például a digitális mosógép vagy mikrohullámú sütő) vagy éppen a járművek egyes elektronikus elemeit. A *kijelzővel és beviteli eszközzel nem ellátott* (de nem *mikrovezérlőt*, hanem *mikroprocesszort* tartalmazó) *platformok* mellett *a mikrovezérlőket* nevezik *beágyazott (embedded)* platformoknak; az ezekre történő fejlesztést nevezik együttesen *beágyazott fejlesztésnek (embedded development).*

***oldalági támadás (side-channel attack) [információ-biztonság]***: olyan, jellemzően az információ védettségét, rejtjelezését szolgáló technikai megoldás elleni támadási forma, amely *nem* maga a technikai megoldás megvalósításában rejlő valamely hibát vagy korlátozást használja ki, hanem *az azt üzemeltető, ahhoz kapcsolódó* valamely rendszerfunkció vagy -elem ellen intéz támadást, ezáltal *közvetett módon* befolyásolva, gyakran *megzavarva* a védelmi megoldás működését. Tipikusan ilyenek például az *adattitkosítás* területén a különböző *véletlenszám-generátorok* hibáit, alacsony entrópiáját kihasználó támadások.

***perzisztencia***: az adott adat *tartós tárolása* (pontosabban *a maradandósága, mint tulajdonság*). Perzisztencia-réteg az alkalmazás valamely rétegének vagy komponensének a neve, amely elem azért felelős, hogy a – jellemzően *objektumokban* – az *operatív,* tehát *illékony memóriában* (*RAM*) tárolt adatok *átkerüljenek* a *háttértárra* (amely lehet merevlemez, nem illékony (*flash*) memória, vagy újabban távoli adattár (*amely jellemzően ugyanezen technológiákat használja, a kliens elől elrejtve*). A gyakorlatban ez az adatbázis-szerver részére történő létrehozó (*create*), frissítő (*update*) vagy törlő (*delete*) parancs kiadását jelenti. A legtöbbször a perzisztenciához sorolják az ilyen adat ezen tárolóból a memóriába történő visszaolvasását (*read*) is, amelynek adatbázisoldalon *lekérdezés (query)* a megnevezése.

***piaci validáció***: valamely szoftverkomponensnek, technológiának a piac általi *elfogadása*, elterjedése. Manapság jellemzően azt mutatja, hogy egyáltalán alkalmas a komponens vagy technológia valamely, meghirdetett feladat ellátására *megfelelő (vö.: kiváló)* minőségben, miközben felhasználása támogatja a *költségkényszernek* való megfelelést és/vagy a *time-to-market* minimalizálására való törekvést. Az erre történő támaszkodás óriási kockázatot rejt, ha az eredeti (*validált*) célkitűzéstől a tervezett, igényelt használat célkitűzései eltérnek. (Ilyen eltérés a komponenst, technológiát felhasználó termék piacra kerülése után, későbbi igény megvalósítása során is felmerülhet.)

***próbapéldány (mock object)***: olyan osztály, amely *interface* szinten megegyezik a *valódi* osztállyal, amelyet *modellez, imitál*, azonban a belső funkcionalitásuk nem

azonos, pontosabban a *próbapéldány* nem valósítja meg az *valódi osztály* funkcionalitását. Elsődleges célja a *unit tesztelés* során bemenő paraméterek biztosítása a tesztelt egységnek, illetőleg a kimenő adatok fogadása.

***projektkockázat [minőségbiztosítás]***: annak veszélye, hogy a projekt, illetve a fejlesztési folyamat nem teljesedik be, kerül végrehajtásra, vagyis *nem eredményez terméket*.

***purizmus***: a letisztultságra, *átláthatóságra*, rendezettségre való törekvés a fejlesztés során, úgy a kód, mint a koncepciók szintjén. Valóban kényszeres szintje problémás lehet, azonban ennek megítélése szubjektív, és a teljes hiánya korlátozza a *strukturális fenntarthatóságot*, a felelősségek követhetőségét, szétválaszthatóságát, ilyenformán nem teszi lehetővé többszereplős *agilis módszertan* alkalmazását. Tapasztalatom alapján egy már meglévő kódbázis esetén nehéz üzletileg érvelni mellette, mint általában a *technikai adósság* ledolgozása mellett; a legtöbbször *súlyosabb problémákat követően* allokálódik erőforrás az ilyen feladatokra (amennyiben a súlyosabb probléma nem jár a *strukturális fenntarthatóság* megszűnésével és ezáltal a projekt leépítésével). Érdemes emiatt valamely szinten a kezdetektől betervezni.

***refaktorálás***: a már meglévő és működőképesnek tekintett forráskód *átstrukturálása* optimizációs, átszervezési, ismétlődés- és/vagy *technikaiadósság*-csökkentő szándékkal anélkül, hogy a kód *működése* változna. Komplexebb rendszerek fejlesztésének *természetes és szükségszerű* része, amely azonban nem jár *eladható* funkcionalitás megvalósításával. Fontos, hogy ez kis, követhető lépésekben valósuljon meg, ezáltal valamely bevezetett hiba esetén könnyű a bevezetett hibát észlelni és elhárítani vagy az előző állapotot visszaállítani. A refaktorálás a valóban *legacy* kódbázisok korszerűsítésének (amikor/ahol ez lehetséges) fő eszköze.

***ReST [architektúra]***: egy hypertext (HTTP) alapú, adatcserét lehetővé tevő *architekturális stílus*. A szó legszorosabb értelmében *nem protokoll*, mivel *elvei*, nem *konkrét előírásai* vannak. Jelentése "*Representational State Transfer*" – *megjelenés-orientált állapotátvitel*. Az átvitt formátum nincs meghatározva, ma a leggyakoribb a JSON (JavaScript Online Notation) használata, de az XML, vagy akár a HTML-be ágyazott base64-kódolt bináris formátum is megvalósíthatja az elveket. Két legfontosabb jellemzője, hogy *a HTTP protokoll metódusait* használja (tehát például rekordot törölni *mindenképp* a *DELETE HTTP metódussal* kell, a *GET* pedig *kizárólag lekérdezésre* használható, nem módosíthat semmit); valamint *kötelező* az *elérhető állapotok (előző, következő rekord, index) hiperhivatkozásként való belefoglalása a válaszba*, amely kikötést HATEOAS-nak (Hypermedia as the Engine of Application State) nevezik. A ReST-konform illesztéseket *RESTful*-nak nevezik. *HATEOAS nélkül semmilyen interface nem RESTful, a HTTP metódusok megfelelő használata nélkül pedig végképp nem; tehát attól, hogy egy interface objektumok, kollekciók elérését teszi lehetővé pl. GET és POST kérések által, és JSON formátumban adja vissza a választ, még nem lesz RESTful!* Ez számos félreértéshez és helytelen értelmezéshez vezet.

***singleton (egyke)***: olyan mechanizmus az objektum-orientált programozásban, amely biztosítja, hogy egy adott osztályból legfeljebb egyetlen példány létezhessen.

***SOAP [protokoll]***: olyan, a Microsoft által fejlesztett, a korábbi *XML-RPC* utódjának szánt adatcsere-protokoll, amely standard megvalósításai – a *ReST*-tel ellentétben – kizárólag XML formátumú adatközvetítést tesznek lehetővé. Jelentése "*Simple Object Access Protocol*". Nevéből fakadóan *egyes objektumok*, illetve kollekciók elérését teszi lehetővé.

***soft skillek [társadalomtudomány]***: azon személyes jellemzők összessége, amelyek lehetővé teszik egy adott egyénnek, hogy hatékonyan és harmonikusan működjön együtt másokkal (Oxford értelmező szótár szerint).

***strukturális fenntarthatóság***: annak a valószínűsége, hogy egy adott üzleti vagy gazdasági folyamat az érintett üzleti szereplő(kö)n belül lezajló változás nélkül, az idő előrehaladtával, a körülményekben felmerülő változások közepette képes *megőrizni* hatékonyságát, bevételtermelő és/vagy működőképességét. Jóval szélesebb értelmű fogalom, mint a *versenyképesség*, ugyanis egyfelől nem kizárólag gazdasági szempontok befolyásolják, másfelől pedig a versenyszférán és/vagy helyzeten bármilyen okból kívül eső szereplőkre, folyamatokra is értelmezhető.

***szolgáltatásjegyzék [tervezési minta]***: egy olyan adattár, amely megnevezés alapján több *egyke (singleton)* osztálypéldányt tárol, és ezeket rendelkezésre bocsátja.

***technikai adósság (technical debt)***: az idők során felhalmozott, nem kritikusnak tekintett teendők összessége. Analógiája számvitelben a költségek, ráfordítások passzív időbeli elhatárolása. Tipikus példa, amikor valamit *meg kellene csinálni rendesen*, de ez a határidőig nem lehetséges, ezért *áthidaló megoldás (work-around)* kerül beiktatásra, a mögöttes probléma pedig a *backlog-ban* rögzítésre. De természetesen az akut probléma elhárultával a nyomás csökken, a következőkben lehet az új feladatra koncentrálni, a technikai adósság pedig csak akkumulálódik.. *A kockázatok mérséklése céljából az üzlet felelőssége a technikai adósság ledolgozására erőforrás biztosítása – legalább az alapproblémák fel nem tárt vetületeinek a megismerése erejéig.*

***technológiai stack***: valamely szoftvertermék vagy szolgáltatás fejlesztésénél és üzemeltetésénél igénybe vett *olyan* technológiai platformok és komponensek összessége, amelyek a termék, szolgáltatás *működéséhez* szorosan kapcsolódnak.
Ennek értelmében ide sorolandó például egy külső, harmadik fél által üzemeltetett SMS küldő szolgáltatás is, ha például azt a termék valamely *távoli eljáráshívás (RPC)* interface-en vagy *API*-n keresztül éri el, *és* az a termék funkcióinak a részét képezi; semmiképpen nem tartozik ide ugyanakkor a fejlesztő részére a fejlesztéshez például a *munkáltató által előírt* hardver- és/vagy szoftverkörnyezet azon része, amely *nem kapcsolódik szervesen a termékhez*, ideértve (a termék platformjával szemben) a fejlesztő által esetlegesen használandó *operációs rendszert* és/vagy *integrált fejlesztői környezetet (IDE)*.

***termékkockázat [minőségbiztosítás]***: annak kockázata, hogy a piacra és/vagy átadásra kerülő termék *ügyfél, megrendelő, célközönség által támasztott célkitűzéseinek nem felel meg* (vagyis *a validáláson* nem felel meg).

***time-to-market***: a (mielőbbi) piacra kerülési idő, mint teljesítmény-indikátor; álláspontom szerint túlértékelt, egyfajta kényszernek tekintendő; ezen jelenségben a szellemi tulajdon védelme hatékonyságának a visszaesése is kiemelt szerepet játszik.

***trade-off***: kifejezetten két, valamely tengelyen ellentétes irányban elhelyezkedő célkitűzés közötti, kompromisszumos pozicionálás. Fő jelentősége rámutatni, hogy az alapul vett környezeten belül a tudomány vagy technika állása szerint a két feltüntetett célkitűzés csak egymás rovására, vagyis egyidejűleg nem, valósítható meg. A szoftverfejlesztésben az egyik tényező jellemzően az *idő (futási, fejlesztési, rendelkezésre állási)* [minimalizálása] szokott lenni.

***unit testing (egységtesztelés, unit tesztelés)***: a program komponenseinek *egységenként* történő tesztelése, előre definiált bemenő adatok és az ezekhez kapcsolt elvárások segítségével. Objektum-orientált megközelítés esetén ezek a komponensek jellemzően osztályok (a csak adatot tartalmazó *"plain"* objektumok tesztelése értelmezhetetlen).

***user story [szoftverfejlesztés]***: az *agilis szoftverfejlesztési módszertan* egyik alapvetése , hogy *a megrendelőnek együtt kell dolgoznia a fejlesztővel*, ezáltal valamely részfolyamatnak a kudarca nem kizárólag a fejlesztést végző felelőssége. Ebből következőleg a kommunikáció során alkalmazott nyelvezetnek úgy a *fejlesztő*, mint a *megrendelő* számára egyaránt érthetőnek és egyértelműnek kell lennie. A mozgalom egyik sajátossága, hogy a technokrata nyelvezetű *specifikáció* helyett kvázi *informális* nyelvezetű ún. *user story*-k írják le az egyes funkciók működését, az azzal szembeni elvárásokat, ekként teljesítve ezen követelményt.

***validálás***: a termék *éles tesztje*; annak ellenőrzése, hogy a termék nagyrészt megfelel a felhasználó, illetve a piac igényeinek. A validálás sikertelenségének valószínűsége a *termékkockázat.*

***whitespace***: a dolgozatban használt *számításechnikai* értelemben azon *karakterek* összessége, amelyek a folyó szöveg, karakterláncok tagolására használatosak. Ilyen alapesetben a *szóköz*, a *tabulátor*, valamint az *új sor* karakterek (*Carriage Return – "kocsi vissza", Line Feed – "soremelés"*). Tágabb, *kiadványszerkesztői* értelemben ide sorolható még mindenféle térköz, behúzás, sorköz, oldaltörés, a sorkizárás során keletkező hézagok és minden egyéb, *szándékosan* elhelyezett vagy szabadon hagyott üres hely.

***work-around***: áthidaló megoldás. Jellemzően nem teljes értékű megvalósítás; tekintve, hogy a legtöbbször a határidő szűkössége miatt helyettesíti a *végleges megoldást*, ilyen esetekben vélelmezhető, hogy a probléma teljes terjedelmében nem került modellezésre. Ez *technikai adósságot* termel, amelynek *törlesztése* kritikusnak tekintendő. A másik fő eset, amikor jellemzően harmadik fél által szállított, vagy egyéb okból zártként kezelendő komponens hiányossága kerül *átmeneti* módon orvoslásra. A legtöbbször

ezek sem *teljes értékű* megoldások, de a határidő szűkössége híján a probléma feltárása nem szükségszerűen részleges, és így a sokszor nem a legoptimálisabb megoldás ezen felül további kockázatot nem feltétlenül jelent.

# 9. IRODALOMJEGYZÉK

## 9.1. Felhasznált irodalom

### 9.1.1. Könyv, folyóirat, publikáció


AVGEROU, CH. (2000): Recognising Alternative Rationalities in the Deployment of Information Systems.
*The Electronic Journal of Information Systems in Developing Countries, Vol. 3., Issue 1. (2000. augusztus) pp. 1-15. doi:10.1002/j.1681-4835.2000.tb00021.x*

BARBER, S., MASON, C. (2011): Web Load Testing for Dummies. *Compuware Special Edition – John Wiley & Sons, Inc.*

BECK, K., BEEDLE, M., VAN BENNEKUM, A., COCKBURN, A., CUNNINGHAM, W., FOWLER, M., GRENNING, J., HIGHSMITH, J., HUNT, A., JEFFRIES, R., KERN, J., MARICK, B., MARTIN, R. C., MELLOR, S., SCHWABER, K., SUTHERLAND, J., THOMAS, D. (2001): Manifesto for Agile Software Development
*Hozzáférhető online: http://agilemanifesto.org/ (2019. 11. 29.)*

BECK, K., FOWLER, M. (2002): Planning Extreme Programming. – *Addison-Wesley Professional*

BERNSTEIN, E. S., TURBAN, S. (2018): The impact of the 'open' workspace in human collaboration.
*Philosophical Transactions of the Royal Society B: Biological Sciences doi:10.1098/rstb.2017.0239*

BOEHM, B. W. (1988): A Spiral Model of Software Development and Enhancement. *Computer Vol. 21. Issue 5. pp. 61-72. (1988. május)*

DIJKSTRA, E. W. (1974): On the role of the scientific thought. – Burroughs, Nuenen
*Elérhető átiratban a Texasi Egyetem archívumából.*
*Web: https://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html (2019. 11. 29.)*

FRIEDMAN, M. (1970): The Social Responsibility of Business is to Increase its Profits. *The New York Times Magazine (1970. szeptember 13.)*

GAMMA, E., VLISSIDES, J., HELM, R., JOHNSON, R. (1994): Design Patterns – Elements of Reusable Object-Oriented Software. – *Addison-Wesley Professional*

HEEKS, R., BAILUR, S. (2018): Analyzing E-Government Research: Perspectives, Philosophies, Theories, Methods, and Practice.
*Government Information Quarterly 24(2), pp. 243-265. doi:10.1016/j.giq.2006.06.005*

HEGEL, G. W. F. (1807): The Phenomenology of Spirit. – *Joseph Anton Goebhart Verlag, Bamberg és Würzburg*
*Németről angolra fordította Terry Pinkard 2010-ben. Párhuzamos fordítás, nyomon követhető.*

HERTZ, M., BERGER, E. D. (2005): Quantifying the Performanceof Garbage Collection vs. Explicit Memory Management.
*Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented Programming, Systems, Languages, and Applications, pp. 313-326.*
*doi:10.1145/1094811.1094836*

HOLLÓ E. (2017): A K+F+I helyzete és finanszírozási lehetőségei a KKV szektorban.
*Doktori értekezés, Szent István Egyetem, Gazdálkodás- és Szervezéstudományok Doktori Iskola*

JANES, A., SUCCI, G. (2012): The Dark Side of Agile Software Development.
*Onward! 2012 Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software, pp. 215-228.*
*doi:10.1145/2384592.2384612*

KOCHER, P., GENKIN, D., GRUSS, D., HAAS, W., HAMBURG, M., LIPP, M., MANGARD, S., PRESCHER, T., SCHWARTZ, M., YAROM, Y. (2018): Spectre Attacks: Exploiting Speculative Execution.
*Cryptography and Security (cs.CR) arXiv:1801.01203*

KRASNER, G. E., POPE, S. T. (1988): A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80.
*Journal of Object-Oriented Programming, Vol. 1., Issue 3. pp. 26-49. (1988. augusztus-szeptember)*

LISKIN, O., SINGER, L., SCHNEIDER, K. (2011): Teaching Old Services New Tricks: Adding HATEOAS Support as an Afterthought.
*Proceedings of the Second International Workshop on RESTful Design, WS-REST 2011, Hyderabad, India (2011. 03. 28.)*
*doi:10.1145/1967428.1967432*

LUCAS-CONWELL, F. (2006): Technology Evangelists: A Leadership Survey.
*A 2006. december 04-i SDForum – "Technológiai Vezetés és Evangélizmus a Részvétel Korában" konferenciára.*
*Web: https://www.gri.co/pub/res/pdf/TechEvangelist.pdf*

MARTIN, R. C. (2000): Design Principles and Design Patterns. – Object Mentor (megszűnt)
*Hozzáférhető az Internet Archive-n keresztül:*
*Web:*
*https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf (2019. 11. 29.)*

MAITLAND, I. (2018): Why the Business Case for Diversity Is Wrong.
*16 Geo. J.L. & Pub. Pol'y 731 (2018). Provided by University of Washington Law Library*

MEIER, J. D., FARRE, C., BANSODE, P., BARBER, S., REA, D. (2007): Performance Testing Guidance for Web Applications: Patterns and Practices. – *Microsoft Corporation*

MUNDIM, A., ROSSI, A., STOCCHETTI, A. (2000): SMEs in Global Market: Challenges, Opportunities and Threats.
*Brazilian Electronic Journal of Economics, 2000. 06. 26.*
*RePEc:bej:issued:v:3:y:2000:i:1:mundim*

PHILIP, T., SCHWABE, G., EWUSI-MENSAH, K. (2009): Critical issues of offshore software development failures.
*University of Zurich Open Repository and Archive. ID: ICIS-0448-2009.R1*

POÓR J., KOVÁCS Á., KOLBE T., CSAPÓ I. (2018): Szakemberhiány és munkaerőmegtartás a kulcsmunkakörökben.
*Kérdőíves kutatás, készítette a Szent István Egyetem Menedzsment és HR Kutató Központja a BDO támogatásával és a BKIK közreműködésével.*

PROKSCH, D., STRANZ, W., PINKWART, A., SCHEFCZYK, M. (2016): Risk management in the venture capital industry: Managing risk in portfolio companies.
*The Journal of Entrepreneurial Finance, Vol. 18., Issue 2. pp. 1-33. (2016 nyár)*

REENSKAUG, T. (1979): MVC – Xerox PARC 1978-1979.
*Elérhető a szerző akadémiai honlapján.*
*Web: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html (2018. 11. 29.)*

ROYCE, W. W. (1970): Managing the Development of Large Software Systems.
*Proceedings of the IEEE WESCON, 26, pp. 328-338. (1970. augusztus)*

SCHOLTES, I., MAVRODIEV, P., SCHWEITZER, F. (2016): From Aristotle to Ringelmann: A large-scale analysis of team productivity and coordination in Open Source Software projects.
*Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn, 2016*

STOUTEN, J., ROUSSEAU, D. M., DE CREMER, D. (2018): Successful Organizational Change: Integrating the Management Practice and Scholarly Literatures.
*The Academy of Management Annals 12(2) (2018. április)*
*doi:10.5465/annals.2016.0095*

TOCCI, J. (2009): Geek Cultures: Media and Identity in the Digital Age.
*Publicly Accessible Penn Dissertations 953, University of Pennsylvania ScholarlyCommons*

VIRONEN, H., KAH, S. (2019): Meeting the Challenges of Digitalisation: Implications for Regional and Rural Development.

*EU European Policies Research Centre, University of Strathclyde, Glasgow. ISBN: 978-1-909522-50-3*
*Web: http://www.eprc-strath.eu/public/dam/jcr:d31b925c-c8da-4200-acfb-e27f6f949efb/EPRP%20111_Meeting%20challenges%20of%20digitalisation.pdf (2019. 12. 04.)*

### 9.1.2. Internetes forrás

ABRAHAMI, Y. (2016): Scaling to 100M: MySQL is a Better NoSQL.
*Wix Engineering (utolsó frissítés: 2018. 07. 17.)*
*Web: https://www.wix.engineering/post/scaling-to-100m-mysql-is-a-better-nosql (2019. 11. 30.)*

Agiles.com (2018): Why ERP projects fail – Lidl stops million-dollar SAP project.
*Web: https://agiles.com/en/why-erp-projects-fail/ (2019. 09. 11.)*

Allianz Risk Barometer 2018 – SME business risks.
*https://www.agcs.allianz.com/news-and-insights/expert-risk-articles/risk-barometer-2018-sme-business-risks.html (2019. 10. 27.)*

BABEL (2017): Babel not working with NODE_PATH.
*GitHub Issue Tracker*
*Web: https://github.com/babel/babel/issues/5618 (2019. 11. 29.)*

BOHUS P. (2019): Álompályának tűnt, de fel kellett adniuk.
*Index – Gazdaság (2019. 08. 22.)*
*Web: https://index.hu/gazdasag/2019/08/22/green_fox_programozokepzes_junior_allas_munka/ (2019. 11. 02.)*

BOLTON, D. (2019): The Fall and Rise of Dart, Google's "JavaScript Killer".
*Dice Insights (2019. 03. 27.)*
*Web: https://insights.dice.com/2019/03/27/fall-rise-dart-google-javascript-killer/ (2019. 11. 29.)*

Bureau of Labor Statistics[1] (2018): Fastest Growing Occupations.
*Occupational Outlook Handbook.*
*Web: https://www.bls.gov/ooh/fastest-growing.htm (2019. 11. 02.)*

Bureau of Labor Statistics[2] (2018): Most New Jobs.
*Occupational Outlook Handbook.*
*Web: https://www.bls.gov/ooh/most-new-jobs.htm (2019. 11. 02.)*

COLLINS, K. (2016): How one programmer broke the internet by deleting a tiny piece of code.
*Quartz Online (2016. 03. 27.)*
*Web: https://qz.com/646467/how-one-programmer-broke-the-internet-by-deleting-a-tiny-piece-of-code/ (2019. 12. 01.)*

COUTU, D. (2009): Why Teams Don't Work.

*Harvard Business Review, May 2009 issue.*
*Web: https://hbr.org/2009/05/why-teams-dont-work (2019. 09. 08.)*

EU Digital Economy and Society Index (DESI) Report (2019) – Integration of Digital Technology.
*Web: https://ec.europa.eu/newsroom/dae/document.cfm?doc_id=59979 (2019. 11. 30.)*

European Commission Directorate General for Regional Policy (2002): Guide to Risk Capital Financing in Regional Policy.
*Web:*
*https://ec.europa.eu/regional_policy/archive/newsroom/document/pdf/draft_venture_financing_guide.pdf (2019. 11. 01.)*

European Institute for Gender Equality (2018): Gender equality and digitalisation in the European Union.
*EIGE Online (2018. 10. 11.)*
*Web: https://eige.europa.eu/publications/gender-equality-and-digitalisation-european-union (2019. 11. 04.)*

EUROSTAT: Foreign language skills statistics.
*Web: https://ec.europa.eu/eurostat/statistics-explained/index.php/Foreign_language_skills_statistics (2019. 10. 27.)*

FORD, J. (2018): Most Tech Companies are Going About Diversity All Wrong.
*Entrepreneur.com*
*Web: https://www.entrepreneur.com/article/317289 (2019. 11. 01.)*

GATES, D. (2013): Boeing 787's problems blamed on outsourcing, lack of oversight.
*The Seattle Times Online (2013. 02. 02, frissítve: 2015. 05. 01.)*
*Web: https://www.seattletimes.com/business/boeing-787rsquos-problems-blamed-on-outsourcing-lack-of-oversight/ (2019. 11. 02.)*

GLOGER, M., DEMPSEY, C., MCCAFFREY, M., WERY, R. (2019): PwC's 22nd CEO Survey: Technology Trends Report 2019.
*Web: https://www.pwc.com/gx/en/ceo-survey/2019/Theme-assets/reports/technology-trends-report-2019.pdf (2019. 11. 01.)*

HackerRank Developer Skills Report (2019)
*Insights based on 71 281 developers.*
*Web: https://research.hackerrank.com/developer-skills/2019 (2019. 11. 30.)*

HE, J., KAPLAN, S. (2017): The debate about quotas.
*Gender and the Economy, 2017.*
*Web: https://www.gendereconomy.org/the-debate-about-quotas/ (2019. 11. 01.)*

KLEINBARD, D. (2000): The $1.7 trillion lesson.
*CNN Money – Technology, 2000. 11. 09.*
*Web: https://money.cnn.com/2000/11/09/technology/overview/ (2019. 11. 01.)*

KOVÁCS ZS. (2017): A kérdőívezés 5+1 halálos bűne.
*Medium.com (a szerző a Prezi User Experience kutatási vezetője)*
*Web: https://medium.com/@zsuzsannakovacs/a-kérdőívezés-5-1-halálos-bűne-2d5b8545b59e (2019. 11. 01.)*

KRABBENBORG, G. (2018): Insufficient change management main cause of SAP @ Lidl project failure?
*LinkedIn Pulse*
*Web: https://www.linkedin.com/pulse/insufficient-change-management-main-cause-sap-lild-guus-krabbenborg (2019. 11. 29.)*

KRILL, P. (2017): Oracle doesn't want Java EE any more.
*IDG InfoWorld (2017. 08. 17.)*
*Web: https://www.infoworld.com/article/3217347/oracle-doesnt-want-java-ee-any-more.html (2019. 11. 29.)*

KSH [1] (2016): A kis- és középvállalkozások helyzete hazánkban.
*Web: http://www.ksh.hu/docs/hun/xftp/idoszaki/pdf/kkv16.pdf (2019. 09. 10.)*

KSH [2] (2019): A regisztrált vállalkozások száma létszám-kategóriák szerint – GFO'14.
*Web: https://www.ksh.hu/docs/hun/xstadat/xstadat_evkozi/e_qvd021.html (2019. 09. 10.)*

LINTHICUM, D. (2019): Back to waterfall: When agile and DevOps don't work well.
*TechBeacon, 2019. 09. 05.*
*Web: https://techbeacon.com/app-dev-testing/back-waterfall-when-agile-devops-dont-work-well (2019. 11. 29.)*

LIPPMAN, L. H., RYBERG, R., CARNEY, R., MOORE, K. A. (2015): Workforce Connections: Key "Soft Skills" that Foster Your Success: Toward a Consensus across Fields.
*Child Trends Publication #2015-24, funded by USAID*
*Web: https://www.usaid.gov/sites/default/files/documents/1865/KeySoftSkills.pdf (2019. 11. 29.)*

MNB 4/2019. (IV. 1.) számú ajánlása a közösségi és publikus felhőszolgáltatások igénybevételéről.
*Magyar Nemzeti Bank, 2019. 04. 01.*
*Web: https://www.mnb.hu/letoltes/4-2019-felho.pdf (2019. 12. 04.)*

MITRA, S. (2017): The Death of Indian Outsourcing is Imminent.
*HuffPost (2017. 06. 05.)*
*Web: https://www.huffpost.com/entry/the-death-of-indian-outsourcing-is-imminent_b_59357886e4b06c4693fb770a (2019. 11. 02.)*

Oracle (2008): Deterministic Garbage Collection: Unleash the Power of Java with Oracle JRockit Real Time.

*An Oracle White Paper (2008. augusztus)*
*Web: https://www.oracle.com/a/ocom/docs/oracle-jrockit-real-time-1517310.pdf*

PENENBERG, A. L. (2009): James Hong: 'Whether it was viral or word of mouth, it was always based on the content'.
*An interview with James Hong, Co-Founder of 'HotOrNot'*
*Web: https://www.independent.co.uk/life-style/gadgets-and-tech/features/james-hong-whether-it-was-viral-or-word-of-mouth-it-was-always-based-on-the-content-1817994.html (2019. 11. 01.)*

*PHP P++ FAQ (2019).*
*Web: https://wiki.php.net/pplusplus/faq*

PIGNI, D. (2018): Would you pay the price for Extreme Diversity?
*Management Information eXchange Online (a szerző Tech Talent Strategist a Google-nál).*
*Web: https://www.managementexchange.com/hack/would-you-pay-price-extreme-diversity (2019. 11. 01.)*

React.js Docs: Composition vs Inheritance.
*Web: https://reactjs.org/docs/composition-vs-inheritance.html (2019. 11. 30.)*

ROBINSON, A. (2018): Want to Boost Your Bottom Line? Encourage Your Employees to Work on Side Projects.
*Inc.com, 2018. 03. 12.*
*Web: https://www.inc.com/adam-robinson/google-employees-dedicate-20-percent-of-their-time-to-side-projects-heres-how-it-works.html*

ROBISON, P. (2019): Boeing's 737 Max Software Outsourced to $9-an-Hour Engineers.
*Bloomberg Technology (2019. 06. 28.)*
*Web: https://www.bloomberg.com/news/articles/2019-06-28/boeing-s-737-max-software-outsourced-to-9-an-hour-engineers (2019. 11. 02.)*

SCHAFFRATH, T. (2018): Elwis has left the building: What we can learn from Lidl's ERP failure.
*CSB Food Business Insights: Blog for the Smart Food Production (2018).*
*Web: https://food-blog.csb.com/us-en/elwis-has-left-the-building-what-we-can-learn-from-lidls-erp-failure (2019. 09. 11.)*

SCHRÖDER, CH. (2018): The Challenges of Industry 4.0 for Small and Medium Sized Enterprises.
*Web: http://www.openisme.eu/challenges-industry-4-0-small-medium-sized-enterprises/ (2019. 09. 09.)*

Stack Overflow Developer Survey (2019) – Technology.
*Stack Overflow Insights.*
*Web: https://insights.stackoverflow.com/survey/2019#technology (2019. 11. 29.)*

STEMMLER, K. (2019): Over $85 billion spent on fixing bad code.
*Web: https://khalilstemmler.com/articles/software-professionalism/developer-principles/*

SUDDATH, C. (2008): Brief History of the Crash of 1929.
*TIME Magazine Online, 2008. 10. 29.*
*Web: http://content.time.com/time/nation/article/0,8599,1854569,00.html (2019. 11. 01.)*

The World Bank Group (2018): Stimulating Business Angels in the Czech Republic.
*International Bank for Reconstruction and Development, 2018.*
*Web: http://documents.worldbank.org/curated/en/633181541532716870/pdf/131855-WP-PUBLIC-CRBusinessangelsOctpages.pdf (2019. 12. 04.)*

Third Stage Consulting (2018): Lessons from an SAP Failure at Lidl.
*Web: https://www.thirdstage-consulting.com/lessons-from-an-sap-failure-at-lidl/ (2019. 09. 09.)*

UCLA Center for Mental Health in Schools (2010): About "Nerds" and "Geeks" as an Identified Subculture.
*Web: http://smhp.psych.ucla.edu/pdfdocs/youth/nag.pdf (2019. 11. 01.)*

WEINSTEIN, J.: Top Three Causes of Failure in Coding Bootcamp.
*Web: https://careerkarma.com/blog/three-causes-of-failure-in-coding-bootcamp/ (2019. 11. 02.)*

WHATWG (2017) – Make custom attribute rules consistent with custom element name rules.
*GitHub Issue Tracker*
*Web: https://github.com/whatwg/html/issues/2271 (2019. 11. 29.)*

ZIMRE ZS. (2018): Négy hónap alatt új szakma – de vajon tényleg értékelhető tudást ad? – A programozó gyorstalpalók
*Wmn.hu, 2018. 02. 07.*
*Web: https://wmn.hu/wmn-life/48247-negy-honap-alatt-uj-szakma-de-vajon-tenyleg-ertekelheto-tudast-ad--a-programozo-gyorstalpalok (2019. 11. 02.)*

### 9.1.3. Jogszabály, rendelet, irányelv

2013. évi V. törvény a Polgári Törvénykönyvről (Ptk.)
*2019. 11. 20.-án hatályos állapot.*

2012. évi I. törvény a Munka Törvénykönyvéről (Mt.)
*2019. 11. 20.-án hatályos állapot.*

## 9.2. További releváns irodalom

*Az itt felsorolt kiadványok, cikkek az alkalmazás fejlesztésének és piaci bevezetésének, illetve az elvégzett vagy az elvégezhető esetleges további kutatásnak szempontjából*

*releváns információkat tartalmaznak, feldolgozásukra azonban elsősorban terjedelmi okok miatt nem került sor.*

### 9.2.1. Könyv, folyóirat, publikáció


BARANN, B., HERMANN, A., CORDES, A.-K., CHASIN, F., BECKER, J. (2019): Supporting Digital Transformation in Small and Medium-sized Enterprises: A Procedure Model Involving Publicly Funded Support Units.
*Konferencia-előadás, Proceedings of the 52nd Hawaii International Conference on System Science (Maui, Hawaii, 2019. január 08.-11.)*

DROGSETH, D. N., CHELLAM, S. V. (2018): Putting the Ops into the DevOps. *IBM Special Edition. – John Wiley & Sons, Inc.*

ENISA (2017): eIDAS: Overview on the implementation and uptake of Trust Services.
*Web: https://www.enisa.europa.eu/publications/eidas-overview-on-the-implementation-and-uptake-of-trust-services/at_download/fullReport (2019. 09. 09.)*

ICO (2018): Guide to the General Data Protection Regulation (GDPR).
*Web: https://ico.org.uk/media/for-organisations/guide-to-the-general-data-protection-regulation-gdpr-1-0.pdf (2019. 09. 08.)*

KROSNICK, J. A., PRESSER, S. (2009): Question and Questionnaire Design.
*Marsden, P. V. [szerk.], Wright, J. D. [szerk.] (2010): Handbook of Survey Research, 2. kiadás, pp. 263-344. – Elsevier Publishing.*

O'Reilly (2016): Modern Javascript – A Curated List of Chapters from the O'Reilly Javascript Library. – O'Reilly Media

PLECHAWSKA-WOJCIK, M., MORA, S. L., WOJCIK, L. (2013): Assessment of User Experience with Responsive Web Applications using Expert Method and Cognitive Walkthrough – A Case Study.
*Proceedings of the 15th International Conference on Enterprise Information Systems, pp. 111-118. doi:10.5220/0004443001110118*

SMITH, M. L. (2005): Overcoming theory-practice inconsistencies: Critical realism and information systems research.
*Information and Organization (2006. szeptember)*
*doi:10.1016/j.infoandorg.2005.10.003*

STURGEON, PH. (2016): Build APIs you won't hate. – *Lean Publishing*

SZIRMAI A., SZÉP A. (2018): A nemzetközi pénzügyi beszámolási standardok (IFRS) alapján teljesített statisztikai adatszolgáltatás módszertanának háttere és tapasztalatai.
*Statisztikai Szemle 96. évfolyam 5. szám, pp. 489-521.*

VIJIYAN, G. (2015): Current Trends in Software Engineering Research.
*Review of Computer Engineering Research, Vol. 2., Issue 3. (2015) pp. 65-70. doi:10.18488/journal.76/2015.2.3/76.3.65.70*

### 9.2.2. Internetes forrás

Deloitte Insights (2019): Tech Trends 2019 – Beyond the digital frontier.
*Web:*
*https://www2.deloitte.com/content/dam/Deloitte/br/Documents/technology/DI_TechTrends2019.pdf (2019. 09. 10.)*

Maruti Techlabs (2016): 5 Challenges need to be overcome in Web Application Development.
*Web: https://medium.theuxblog.com/5-challenges-need-to-be-overcome-in-web-application-development-c03a67726ff1 (2019. 09. 09.)*

SCHÖLLHORN, N. (2019): Moving away from magic—or: why I don't want to use Laravel anymore.
*FreeCodeCamp Online (2019. 02. 19.)*
*Web: https://www.freecodecamp.org/news/moving-away-from-magic-or-why-i-dont-want-to-use-laravel-anymore-2ce098c979bd/ (2019. 11. 29.)*

### 9.2.3. Jogszabály, rendelet, irányelv

Az Európai Parlament és a Tanács (EU) 2016/679 rendelete (2016. április 27.) a természetes személyeknek a személyes adatok kezelése tekintetében történő védelméről és az ilyen adatok szabad áramlásáról, valamint a 95/46/EK rendelet hatályon kívül helyezéséről (általános adatvédelmi rendelet)
*Web: https://eur-lex.europa.eu/legal-content/HU/TXT/?uri=celex%3A32016R0679 (2019. 09. 08.)*

# 10. MELLÉKLETEK

## 10.1. I. melléklet: A projektben érintett stakeholder-ek szempontjai, elvárásai

***Ügyfél: döntéshozó***

- határidők betartása
- ígért funkcionalitás specifikáció vagy storyk szerinti működése
- költségkorlát!
- látványos vezetői áttekintő funkciók, hordozható eszközökön is
- értesítések
- szükséges mértéket nem meghaladó adatszolgáltatás
- GDPR (adatvédelmi felelősség átvállalása)

**Ügyfél: felhasználó**

- egyszerű használhatóság
- sok adat mellett gyors működés
- robusztusság
- értesítések

**Fejlesztő, integrátor: döntéshozó**

- integrációs körök számának minimalizálása
- határidő betartatása
- infrastrukturális és üzemeltetési költségek minimalizálása
- esetleges integrációs igények részletes és teljes körű megismerése
- GDPR (adatvédelmi felelősség)

**Fejlesztő, integrátor: kivitelező**

- egyértelmű és teljes körű igények
- kommunikációs lehetőség az ügyféllel (szükség esetén)
- hozzáférés az infrastruktúrához

**Projektmenedzser, projektgazda**

- gördülékeny információ-csere
- domain-specifikus tudás és nyelvezet használata

- megkívánt hozzáállás fejlesztők és ügyfelek részéről

**Kormányzat**

- adatcsere*
- jogszabályoknak való megfelelés
- adatszolgáltatás

**Infrastruktúra-szolgáltató**

- terhelés-minimalizálás
- *abuse compliance*

## 10.2. II. melléklet: A kérdőív összeállításának szempontjai

Az előtesztelés során műszaki vagy szerkezeti probléma, *félreérthető kérdés* nem került feltárásra; néhány elírás és félreérthető fogalmazás került korrekcióra.

A feltett kérdések közül azok, amelyek nem a célcsoport kiválasztását teszik lehetővé (tehát a kvantitatív mérési adatok) *Likert skála* szerint értékelhetőek. Az ezekre adott válaszok *egy kérdésen belül* – a 0, vagyis *nem tudja / nem válaszol* adatok elhagyásával – standardizáltnak tekinthető.

A kitöltés során egyetlen visszajelzés érkezett arra vonatkozóan, hogy egy adott ügyviteli szoftverkomponens használatára vonatkozó opciók *nem tartalmaznak számára megfelelő válaszlehetőséget*. De mivel a kérdőív az én irányomban *valóban teljesen anonim*, elérhetőséget pedig az illető nem adott meg (illetve "nem válaszol" lehetőség is adott), így ezt a megközelítést nem sikerült maradéktalanul feltárni.

Egy kivétellel az összes kérdés alapvetően *zárt*, és a kérdések mindegyike *direkt*.
A minta várható nagysága, a kitöltéssel elérhető haszon (nemléte), illetve a tény, hogy az adott válaszok a későbbi kérdéseket egyáltalán nem befolyásolják, azt eredményezték, hogy az esetleges *összejátszás* lehetőségét elvetettem, így *randomizálást* nem végeztem.

A kérdőív kérdései megtalálhatóak a *III. mellékletben*.

A kérdőív 6 részre van tagolva (6 "oldal"), és 20 kérdést tartalmaz. Ezek többsége (a bevezető, és az utolsó oldal kivételével) mátrix-kérdés, tehát jóval több adat áttekintését igényli, ugyanakkor lehetőséget biztosít a gyors kitöltésre.
A legelső kérdés kivételével mindegyik kérdés tartalmaz *nem tudja/nem válaszol* értelmű lehetőséget, amit azonban *explicit* kell kiválasztani.

A nem mátrix jellegű kérdések mindegyikénél van "*egyéb*" lehetőség; ezek a lehetőségek statisztikailag nem kerülnek kiértékelésre, céljuk elsősorban a kényszerválaszok torzító hatásának, vagy az ez okozta esetleges lemorzsolódásnak a csökkentése.

A legutolsó kérdés egy szabad szöveges vélemény kifejezésére alkalmas szöveg. Statisztikailag nyilván ez sem értelmezhető, azonban saját példámon tudom, hogy egy kérdőív kitöltése *minden esetben* kompromisszumokkal jár; ezek ténye nem, csak mértéke lehet kérdéses a legnagyobb körültekintés esetén is; és célszerű lehetőséget biztosítani a különböző észrevételek, fenntartások közlésére, akárcsak a jövőbeni módszertani fejlesztés céljából is.

A kérdőív kitöltése közben az előző oldalra visszalépni *nem lehet*.

A "nulladik" oldal nem számozott. Ezen a legelső kérdés szegmentálja a kitöltőket.
Az oldal további kérdései demográfiai jellegűek (nem, életkor, végzettség, illetve munkahely szerint). *Ezeknek előzetes jelentőséget nem szántam*; hasznos lehet viszont a fejlesztés későbbi elemeinek ütemezése szempontjából (például meg lehet vizsgálni, hogy az 50 év fölötti vezetők szemszögéből van-e relevanciája mondjuk egy natív "áttekintő" okostelefon-alkalmazás fejlesztésének egyéb teendők elé sorolására). Az életkor sávos lehetőség, nem konkrét szám megadását biztosítja; azt pedig szándékoltan kerültem, hogy a kitöltő születési évét kérjem be, ez ugyanis véleményem szerint *szubjektíven kellemetlen érzést kelthet, valamint személyiségi jogi aggályokat is felvet*.

Az ezt követő oldalon kezdődik a számozás. Ez várakozásaim szerint segít csökkenteni a pszichés időveszteség-érzetet.

Az első érdemi oldal mátrixban vizsgálja a különböző vállalati ügyviteli szoftverkomponensek *használatának tényét* a múltban, a jelenben, illetve a *szándékát* a jövőben, majd rögtön az ezt követő oldalon egy megtévesztésig hasonló kérdéssor az ezekkel kapcsolatos *tapasztalatot, élményt* méri *Likert skálán*, amely a tervezett közömbösség kiküszöbölésének érdekében *négyfokozatú*.

A harmadik oldalon 15, véleményem szerint egy hasonló termékkel/szolgáltatással szemben adott esetben elvárható *szempont* van felsorolva, ezeket lehet ugyancsak *négyfokozatú Likert skálán* értékelni.

A kérdőív legfontosabb része ezzel véget ért. További két oldalon *szentimentum-vizsgálatra* kerül sor. Itt már *ötfokozatú (szintén Likert) skálán* lehet a felkínált állításokkal egyetérteni, vagy azokat elutasítani. Ez lehetővé teszi a "közömbösséget" ("egyet is ért, meg nem is") azon látogatóknak, akik ezen kérdéseket esetleg *offenzívnek* találnák. Ennek megvan a reális esélye, ugyanis ezen oldalak kérdései *szándékosan elfogultak.*

Mindezek miatt stratégiai jelentősége van annak, hogy a fejlesztés irányvonalának meghatározásához szükséges kérdések *ezen rész előtt* kerültek megválaszolásra; illetve annak is, hogy *a kérdőívben nem lehet visszalépni, választ módosítani.*

Ezen elfogultság azonban *bár szuggesztív*, mégsem *öncélú*: egyes kérdéseknél a saját véleményem irányába *hajlik* a megfogalmazás, másoknál az azzal éppen ellentétesnek tekinthető irányba; egy oldallal később pedig ellentétesnek tekinthető, de hasonlóan *megosztó módon* megfogalmazott válaszpárok közül választhat a kitöltő.

Ez a célkitűzés alapján lehetőséget biztosít arra, hogy *pszichológiai* következtetéseket vonjak le változatos szempontok szerint: például kiszűrhetem azokat, akik kifejezetten passzív-aggresszívak, vagy meg tudom különböztetni a *minden változásnak következetesen ellenállókat* azoktól, akiknek *jogosnak vélt aggályuk* van valamely *evolúciós/futurista* gondolatkörről (amivel, mint az előző fejezetben kiemeltem, abszolút nem lennének egyedül).

Hogy erre sort kerítek-e, azt a válaszadók és a válaszok eloszlása dönti majd el; előzetesen az vélelmezhető, hogy valamely két szempont közötti *szoros eredmény* esetén vélhetően figyelembe veszek hasonló szempontokat, míg látványosan elkülönülő eredmények esetén esetleg eltekintek ettől.

Fontos még megemlíteni, hogy valamely *megosztó, vagy elfogult válasz-megfogalmazás* semmiképpen sem lehet olyan, amelyet *inventívnek* lehetne tekinteni; ezek mindegyike hitelesnek (de nem *elsősorban* tudományosnak) tekintett nemzetközi online kiadványok által is közzétett híreken és/vagy korábbi kutatásokban már feltárt félelmeken alapul.

## 10.3. III. melléklet: A kérdőív kérdései

A kérdőív kérdései közül az alábbiakban csak a kutatásban kiértékelt kérdések vannak felsorolva. Néhány kérdés, amely esetlegesen ellentmondásos eredmények tisztázására, vagy egy esetleges, irodalmilag nem alátámasztott pozitív vagy negatív trend hátterének vizsgálatára került be a kérdőívbe, ilyen ellentmondások, jelenségek híján kiértékelésre nem került, ezért itt sem szerepel.

**1. Melyik jellemző a leginkább az Ön professzionális szerepkörére?**

- Vezető / Döntéshozó
- Adminisztratív / Ügyintéző / Felhasználó
- Szakmai érdeklődő (Információ-technológia, Ügyvitel)
- Egyik felsorolt csoportba sem tartozom

**2. Az Ön neme** [ Nő; Férfi; Nem válaszol; Egyéb ]

**3. Az Ön életkora** [ 18 éven alul; 18-29 év; 30-39 év; 40-49 év; 50-59 év; 60 év fölött; Nem válaszol ]

**4. Az Ön lakóhelye (tartózkodási helye)** [ Budapest; Megyeszékhely, megyei jogú város; Város; Nagyközség; Kistelepülés; Magyarországon kívüli; Nem válaszol ]

**5. Az Ön legmagasabb befejezett iskolai végzettsége** [ 8 általános; Szakiskola; Szakközépiskola / Szakgimnáziumi érettségi; Gimnáziumi érettségi; FOSZK / Technikum / Felsőfokú OKJ; Főiskolai alapfokozat (BA/BSc); Egyetemi alapfokozat (BA/BSc); Osztatlan (Jogi, Orvosi) / Mesterfokozat (MA, MSc, MBA); PhD; Nem válaszol ]

**6. Az Ön munkahelye / tevékenységének helyszíne** [ Mikrovállalkozás (10 fő alatt); Magyar tulajdonú kisvállalkozás (10-49 fő); Külföldi tulajdonú kisvállalkozás (10-49

fő); Magyar tulajdonú középvállalkozás (50-249 fő); Külföldi tulajdonú középvállalkozás (50-249 fő); Magyar tulajdonú nagyvállalat (250 fő fölött); Multinacionális nagyvállalat (250 fő fölött); Állami tulajdonú vállalat; Oktatási intézmény; Közigazgatási, közhatalmi intézmény, jogi személy; Civil szervezet, közhasznú társaság; Egyéni vállalkozó, szabadúszó; Egyéb; Nem válaszol ]

**7. Használ-e, használt-e Ön valamilyen elektronikus ügyviteli, logisztikai, és/vagy kereskedelmi megoldást az alábbiak közül (internetes/felhős, intranetes, vagy telepített)?** [ Igen, használok jelenleg is; Igen, használtam korábban; Igen, korábban használtam, de a jövőben nem tervezem; Sosem használtam, de tervezem; Sosem használtam, nem is szeretnék; Nem válaszolok ]

- Számlázó rendszer
- Raktárkészlet-kezelő / logisztikai rendszer
- CRM rendszer
- Bérszámfejtő rendszer
- Munkaidő-nyilvántartó rendszer
- Helpdesk (ügyfélszolgálati / -támogatási) rendszer
- Online marketing (hírlevélküldő, konverziónövelő, érdeklődéskezelő) rendszer
- Webáruház (értékesítésre, nem vevőként)
- Komplex termelés-/folyamatirányító rendszer (ERP) (pl. SAP Business One)

**8. Mennyire elégedett Ön általánosságban véve az Ön által ismert / használt / Önnek ajánlott rendszerek alábbi tulajdonságaival?** [ 5 fokozatú Likert skála: 5 = Teljes mértékben elégedett; 1 = Abszolút elégedetlen; emellett *Nem válaszol* lehetőség biztosított ]

- Funkciók teljeskörűsége (mindent tartalmaz, amit Ön/az Ön cége igényel)
- Új funkció bevezetése, megvalósítása (az Ön, Önök igényei szerint)
- Megbízhatóság, hibamentesség
- Sebesség, teljesítmény
- Kezelőfelület felhasználóbarátsága (az ismétlődő feladatok gyors kivitelezhetősége)
- Kezelőfelület modernsége, esztétikája
- Mobileszközökön (elsősorban telefonon) való használhatóság
- Ügyfélszolgálat hozzáférhetősége, elérhetősége
- Ügyfélszolgálat segítőkészsége
- Ügyfélszolgálat felkészültsége

**9. Általánosságban véve mennyire tartja Ön fontosnak ezeket a tulajdonságokat, amennyiben Ön használna egy hasonló rendszert?** [ 5 fokozatú *Likert skála*: 5 = Teljes mértékben elégedett; 1 = Abszolút elégedetlen; emellett *Nem válaszol* lehetőség biztosított ]

- Funkciók teljeskörűsége (mindent tartalmaz, amit Ön/az Ön cége igényel)
- Új funkció bevezetése, megvalósítása (az Ön, Önök igényei szerint)
- Megbízhatóság, hibamentesség
- Sebesség, teljesítmény
- Kezelőfelület felhasználóbarátsága (az ismétlődő feladatok gyors kivitelezhetősége)
- Kezelőfelület modernsége, esztétikája
- Mobileszközökön (elsősorban telefonon) való használhatóság
- Ügyfélszolgálat hozzáférhetősége, elérhetősége
- Ügyfélszolgálat segítőkészsége
- Ügyfélszolgálat felkészültsége
- Minél több funkció (számlázás, raktárkészlet-kezelés, ügyfélkezelés ...) egy helyen és módon történő kezelése
- Felhő alapú működés (üzemeltetési költség, munkaerő-igény nélkül)
- Telepíthető megoldás (a tárolt adatok ellenőrizhetősége)
- Könyvelés, mérlegkészítés támogatása
- Integrációs lehetőség a meglévő rendszerekkel (számlázó, raktárkészlet-kezelő, ügyfélkapcsolati, ...)

**10. (Ön szerint) Általánosságban véve a legtöbb mai weboldal...** [ 5 fokozatú *Likert skála*: 5 = Teljes mértékben egyetért; 1 = Abszolút nem ért egyet; emellett *Nem válaszol* lehetőség biztosított ]

- Tele van hibákkal
- Átgondolatlan szerkezetű, elrendezésű
- "Kémkedik" a látogatók után
- Lassú, túlterheli a számítógépem, mobiltelefonom
- Sokkal jobb, mint néhány éve volt

**11. (Ön szerint) A legtöbb mai üzleti szoftver...** [ 5 fokozatú *Likert skála*: 5 = Teljes mértékben egyetért; 1 = Abszolút nem ért egyet; emellett *Nem válaszol* lehetőség biztosított ]

- Tele van hibákkal
- Abszolút nem felhasználóbarát
- Fejlesztője vélhetően saját maga számára fejleszti a terméket, a felhasználók véleményének figyelembe vétele nélkül
- Kinézete leragadt a 2000-es évek elején
- Fejlesztői abszolút nem érdekeltek a felhasználói elégedettség fokozásában
- Kinézete folyamatosan változik, átalakul, követhetetlen
- Remek! Örülhetünk, hogy sikerült végre ide eljutnunk

**12. (Ön szerint) Az otthonokba szerelt digitális mérőórák szolgáltatói megfigyelése ("okos mérés") inkább:**

- remek lehetőség az energiafelhasználás hatékonyságának javítására
- a szolgáltatóknak segít a fogyasztók szokásairól adatokat gyűjteni, amelyeket később értékesítenek
- *nem válaszol*

**13. (Ön szerint) A növekvő mértékű elektronikus adóhatósági adatszolgáltatás inkább:**

- a gazdaság "kifehérítését" segíti az adóelkerülők féken tartásával
- a bennfentes piaci szereplőket segíti, hogy érzékeny információkhoz jussanak a kevésbé tőkeerős vagy beágyazott vállalkozások üzleti tevékenységéről
- *nem válaszol*

**14. A felhő alapú adattárolás véleményem szerint:**

- teljesen biztonságos és költséghatékony
- számos kockázatot rejt magában
- *nem válaszol*

**15. A céges adatok sokkal nagyobb biztonságban vannak egy:**

- itthoni szerveren
- külföldi szerveren
- *nem válaszol*

**16. Ha megtehetem, inkább:**

- az itthoni gazdaságot támogatom a versenyképessége javítása céljából
- a legmodernebb technológiára költenék, minden más mellékes

- *nem válaszol*

## 10.4. IV. melléklet: A teljesítménymérési környezet részletes leírása

### *Szempontok*

A teszt-eset egy kellően leegyszerűsített üzleti adathalmaz lekérdezését és HTTP-n keresztüli visszaadását teszi lehetővé.

Amennyire korlátozott használhatóságú egy, például üres adathalmazt (hacsak nem éppen a *technológiai stack* minimális válaszidejét, késleltetését akarjuk vizsgálni) visszaadó teszt-eset, annyira célszerűtlen az is, hogy valamely nagyon specifikus, és ehhez mérten bonyolult folyamatot modellezzünk, amelyhez még csak hasonló sem fog előfordulni a megcélzott szegmens használati esetei 99.9 százalékában.

A komparatív tesztek fő szempontja a *minél nagyobb azonosság* a tesztelt technológiák, platformok teszt-esetei között, a *csereszabatos*, *de rosszul megválasztott elemek torzító hatását kiküszöbölendő*. Emiatt *minden* eset ugyanazt az adatbázist, azonos rekordszámmal fogja használni.

### *Adatbázis*

Az adatbázis létrehozása egy előre megalkotott SQL szkripttel történik, feltöltése pedig részben szintén előre definiáltan, másodrészt *tesztadatok* felhasználásával, végezetül *a tesztadatok* véletlenszerű *szétszórásával*, amely egy PHP inicializáló (*seed*) szkripttel kerül megvalósításra.

A tesztadatok (termék) létrehozásában a *Mockaroo* (www.mockaroo.com) címen elérhető kiváló szolgáltatás volt segítségemre. Az előre megadott séma alapján generálható sorok maximális száma 1000, így 1000 termék került az adatbázisban elhelyezésre. :)

### *Tesztelt technológiák*

Az összehasonlításban szereplő technológiák "telepítése" során az általuk biztosított legegyszerűbb telepítési megoldás került használatra, kivéve azt, ahol ez valamilyen példaalkalmazás letöltését és átalakítását jelentette volna. Ennek a relevanciája, hogy akár az adott platformmal most ismerkedő, akár valamely már meglévő, *legacy* alkalmazást modernizálni kívánó érdeklődő nagy eséllyel így fog elindulni annak használata során. Ezen példaalkalmazás mellőzése fontos szempont volt, megelőzendő, hogy valamely, a platformon jelen mérés szempontjából *egyébként mellőzhető*, de az alapul vett példában szereplő funkció *inicializációs ideje* torzítsa a mérési eredményeket. (Ezzel szemben a keretrendszerben alapból szereplő, de nem használt funkciók eltávolításra *nem* kerültek; hasonlóan ahhoz, ahogyan az érdeklődők sem itt fogják kezdeni az összehasonlítást vagy a *zöldmezős* fejlesztést.)

A teszt-eset során az adatok lekérdezése *minden platformon két különböző módon* történik.

Az első esetben (***A***) kizárólag a *terméktörzs* ***teljes*** lekérdezése és visszaadása zajlik szabványos JSON formátumban; ennek az összehasonlítása a mindennapokban kevéssé releváns, viszont támpontot adhat egy esetleges *mikroszolgáltatás alapú platform* összeadódó minimális késleltetési idejének a felméréséhez.

A második eset (***B***) egy *komplexebb*, aggregált adatot *és* kapcsolódó adatokat is lekérdező eset. Itt *élesen* elkülönül a különböző megközelítések *(ORM, lekérdezés, domain alapú megközelítés)* közötti teljesítménybeli eltérés. *Itt is az* ***összes****, 1000 rekord és a kapcsolódó adatok lekérdezése történik.*

A tesztelt platformok között szerepel a Java legelterjedtebb, SpringBoot 4 platformja, a Hibernate JPA megvalósítást használva (tehát ORM használatával), PHP-n a Symfony 5 keretrendszer külön-külön Doctrine ORM-el és a DBAL-on keresztül közvetlen SQL lekérdezésekkel, a Laravel 5.8 keretrendszer a saját Eloquent ORM-évelés SQL lekérdezés-összeállítójával (Query Builder), valamint a saját keretrendszerem közvetlen SQL lekérdezésekkel.

#### *Hardver- és szoftverkörnyezet*

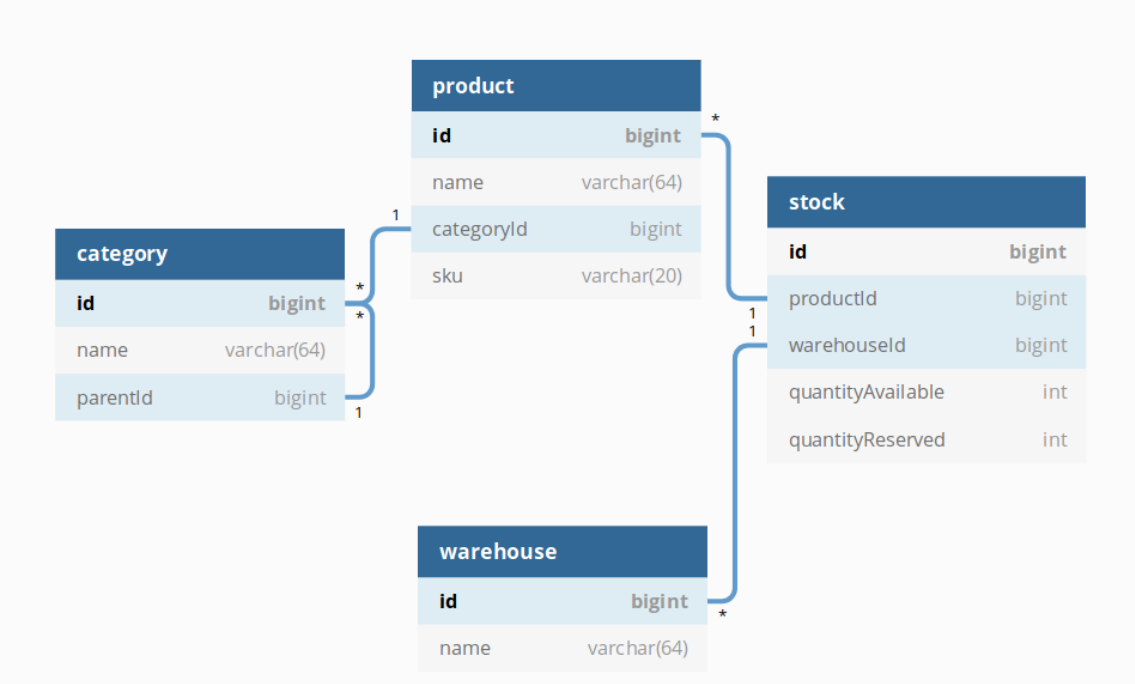
A tesztgép egy több, mint 10 éves technológiára épülő 4 magos Xeon platform (E5450) 12 gigabyte DDR3 memóriával és 256 MB SSD-vel. A *komparatív* módszer miatt a pontos hardver-specifikációnak nincs jelentősége, inkább csak a mért adatok kontextusba helyezhetősége miatt érdekes.

A szoftverkörnyezet Java oldalon Java EE 8 (Apache Tomcat/9.0.27); PHP oldalon PHP-FPM 7.3.12. Az adatbázis egységesen a MySQL 5.7-re épülő MariaDB 10.2.18; mindegyik platformhoz egy *nginx* (1.14.2 verzió) továbbítja a bejövő HTTP kéréseket (*fordított / edge side proxy* módban).

A teszteléshez egy egyszerű szkriptet írtam, amely az Apache Foundation ApacheBench nevű, egyebek mellett alapszintű terheléses teszt végzésére alkalmas parancssori segédprogramját hívja meg. A konkurrens szálkezelés sajátosságai és platformonként eltérő mivolta miatt a *párhuzamos* kérések számát négyre (magonként egy) korlátoztam; a tesztenkénti kérések száma összesen 1000 darab. A szkript az egyes platformok tesztelését követően öt másodpercig várakozik (lemezműveletek, memória-mozgatás, lapozás stb. esetleges befejezésére) az új teszt megkezdése előtt. Ez lehetővé teszi vizuálisan az erőforrás-figyelőn az arról való megbizonyosodást, hogy valamely tesztsorozat lezajlása után nem marad futó háttérfolyamat/-terhelés a következő teszt eredményét befolyásolandó (vagyis a processzorhasználat-érték "leesik" nullára). A *13. ábrán* látható, hogy az 1000 kérés hozzávetőlegesen éppen optimális a tesztplatformra; a kéréstartomány vége felé, az "optimális" fölötti erőforrás-kihasználás mellett megjelenik a legtöbb platformnál a válaszidő *ugrásszerű* növekedése.

### *Adatszerkezet*

A megvalósított táblaszerkezetek egység-kapcsolat diagramját a *14. ábra* szemlélteti. A lehető legegyszerűbb, valós igényt modellező szerkezetre törekedtem, amely azonban mindenképpen tartalmaz olyan lekérdezési lehetőséget, amely 1:1 kapcsolattal *nem ábrázolható*.




*14. ábra: a teszt-eset adatbázis-modellje*

*Saját grafika (készült a dbdiagram.io szolgáltatással)*

A táblák feltöltésekor 1000 darab termék került feltöltésre, szétszórva 4 darab, harmadik szintű mélységben elhelyezett kategória között. *Technológiai okokból a **B** típusú, aggregált listázásnál a kategóriák rekurzív lekérdezése nem történik meg; kizárólag a termék konkrét (al)kategóriája kerül visszaadásra.*

A tervek szerint a teszt reprodukciójához szükséges adatokat, kódokat és leírásokat az Interneten a *https://www.strongholdmedia.hu/webbm* címen elérhetővé teszem a licencelési és adatbiztonsági kérdéseket, illetve a verziókezelési nehézségek áthidalását követően. Ez reményeim szerint legkésőbb a következő év első napjaiban megtörténik.