

TDK-DOLGOZAT

Hangya Viktor

Koós Krisztián

Szegedi Tudományegyetem
Természettudományi és Informatikai Kar
Szoftverfejlesztés tanszék

Szoftverfejlesztők munkájának minősítése

Készítette:

Hangya Viktor
programtervező informatikus Bsc.
III. évfolyam

Témavezető:

Dr. Ferenc Rudolf
egyetemi adjunktus

Külső konzulens:

Koós Krisztián
programtervező informatikus Bsc.
III. évfolyam

Bakota Tibor
FrontEndART Szoftver Kft.

Szeged
2012

Tartalomjegyzék

<u>1 Bevezetés.....</u>	<u>4</u>
<u>2 Minőségbiztosítás.....</u>	<u>7</u>
<u>2.1 Forráskód metrikák.....</u>	<u>7</u>
<u>2.2 Verziókezelő rendszerből kinyerhető információk.....</u>	<u>9</u>
<u>2.3 Minősítő modell.....</u>	<u>10</u>
<u>3 Fejlesztői minősítő modell.....</u>	<u>13</u>
<u>3.1 Szenzor pontok.....</u>	<u>13</u>
<u>3.2 Aggregált pontok.....</u>	<u>18</u>
<u>4 Megvalósítás.....</u>	<u>21</u>
<u>5 Eredmények.....</u>	<u>24</u>
<u>5.1 Elemzett rendszer.....</u>	<u>24</u>
<u>5.2 Gyűjtött adatok.....</u>	<u>25</u>
<u>5.3 Kiértékelés.....</u>	<u>28</u>
<u>6 Kapcsolódó munkák.....</u>	<u>34</u>
<u>7 Összefoglaló.....</u>	<u>36</u>
<u>8 Irodalomjegyzék.....</u>	<u>37</u>
<u>9 Köszönetnyilvánítás.....</u>	<u>39</u>

1 Bevezetés

Egy szoftver fejlesztése során nagyon fontos tényező a fejlesztésre fordított költségek alacsonyan tartása annak érdekében hogy a projekt kifizetődő és nyereséges legyen. Ezért a kezdetektől fogva ügyelni kell minőséges kód előállítására és a különböző hibák alacsonyan tartására. A fejlesztés kezdeti szakaszaiban kisebb erőforrással lehet javítani az előforduló rendellenességeket, mivel ilyenkor még kevés egymástól függő komponenssel rendelkezik az adott rendszer. Később a rendszer bonyolultságából adódóan az egyes hibajavítások más szoftverkomponenseket is érinthetnek, így a javítás költségei sokkal nagyobbak lehetnek. Abban az esetben ha egy telepített rendszerben kell javításokat végezni, a költségek az előbbiek többszöröse lehet, mivel ilyenkor a javításon kívül számos más tevékenységre is szükség lehet, mint például a létrehozott adatok migrációja a javított rendszerbe, szolgáltatás leállítása az új rendszer üzembe helyezéséig, de akár különböző kártérítési díjak kifizetésére is kötelezhet az ügyfél.

Ellentétben más iparágakkal, ahol a projektek költségeinek nagyobb részét a szükséges berendezések és anyagok teszik ki, mint pl. gyógyszer- és építőipar, a szoftverfejlesztésben tipikusan a fejlesztők munkabére a legdrágább. Mindig felmerül a kérdés, hogy ez az összeg jól van-e elkölthetve. Érdemes tehát optimalizálni ezt a költséget, mert a legnagyobb összegből takarítható meg a legtöbb.

A szoftver minőségének becslésére több módszer is alkalmazható. Egyik ilyen a minél átfedőbb tesztek létrehozása és azok elvégzése, mely segítségével megállapítható hogy a rendszer az elvárt funkciókat milyen mértékben teljesíti. A szoftver tesztelését az ISO/IEC 29119 standard [1] definiálja részletesen. Egy másik lehetőség a forráskódok statikus elemzése, mely segítségével olyan rossz előjelekre lehet bukkanni, melyek akár az aktuális állapotban vagy a későbbiekben hibát okozhatnak. Ilyenek például a klónok, melyek egy a rendszerben már létező kódrészlet lemásolásával keletkezhetnek. Abban az esetben okoznak gondot, ha a kódrészlet később módosulásra szorul, viszont csak az eredeti példányt módosítják, a másolatokat nem, így azok hibásan élnek tovább. Ezek a módszerek azonban a szoftver aktuális állapotát képesek becsülni, a fejlesztés későbbi szakaszaira nem adnak információkat.

A szoftver minőségét leginkább befolyásoló tényezők az azt létrehozó fejlesztők. Abban az esetben ha rendelkezünk bizonyos információkkal a szoftverfejlesztők kompetenciáiról,

lehetőség van becslések készítésére az általuk aktuálisan fejlesztett rendszerek minőségét illetően. A fejlesztőket számos szempont alapján lehet értékelni, mely értékeket a becslések során fel lehet használni. Ilyen például a fejlesztő által létrehozott forráskód minőségének elemzése a fent említett módszerek segítségével.

Dolgozatunk egyik célja szoftverfejlesztők olyan alapvető tulajdonságainak felkutatása, melyek hatással lehetnek teljesítményükre és amelyek közvetlenül mérhetők. Ezeket az alap tulajdonságokat különböző módon csoportosítva absztraktabb tulajdonságok hozhatók létre, melyek választ adhatnak olyan kérdésekre, melyek a fejlesztőkkel kapcsolatban közvetlenül nem mérhetők. Ilyen például az a kérdés, hogy mekkora teljesítményt várhatunk el egy adott fejlesztőtől, melyet számos tényező befolyásol. Az elvárásunkra hatással lehet az adott személy képzettsége, gyakorlata, motiváltsága stb. Ezen tulajdonságok kutatása során számos akadályba ütköztünk. Egyrészt igyekeztünk minél több olyan tulajdonságot találni, melyek algoritmikus módon számíthatóak az egyes fejlesztők múltbéli munkái alapján. Egy adott személy előző munkáinak beazonosítására verziókövető rendszer információit használtuk, ezen munkák minőségének elemzésére pedig erre a célra kifejlesztett eszközöket. Így a forráskód minősítésével az azt létrehozó fejlesztőt is minősíthetjük. Azonban nehéz feladat megfelelő tulajdonságok felkutatása, mivel számos esetben a kinyert információk bizonytalanok voltak. Például vegyük azt az esetet amikor az adott fejlesztő eltünteti egy forráskód másolatot. Nem tudni biztosan hogy szándékosan javította ki, vagy csak módosítást végzett a másolaton és így azt tudtán kívül olyan állapotba hozta, hogy nem egyezett meg teljesen az eredeti kódrészlettel. Másfelől az algoritmikusan nem mérhető tulajdonságok esetében is számos tényezőt figyelembe kellett venni. Csak olyan tulajdonságok jöhetnek szóba, melyeket az adott fejlesztőről valaki, akár önmaga vagy felettesei, meg tudnak állapítani. Ezenkívül figyelni kell az adatok kényességére is, néhány esetben nem biztos hogy az egyes személyek megosztanák a valós véleményeiket, attól félve, hogy esetleg negatív következményei lennének azoknak.

A megfelelő tulajdonságokat felhasználva ezután egy minősítő modellt (melyről később bővebben is lesz szó) hoztunk létre, mely az egyes alapvető tulajdonságokat elvontabbakká alakítja, és kiszámítja az értékeiket. Munkánk során néhány már meglévő alkalmazást egészítettünk ki saját szükségleteink alapján, illetve saját segédalkalmazásokat is implementáltunk. A fejlesztőkkel kapcsolatos adatok és a minősítő modell absztrakt tulajdonságainak kiszámítása után megkaptuk, hogy az egyes fejlesztők bizonyos tényezők

alapján az átlaghoz viszonyítva hol helyezkednek el. Ezen értékek segítségével olyan kérdésekre tudunk választ adni, mint például az egyes fejlesztők mely tulajdonságai gyengébbek és azokat hogyan lehetne javítani.

A dolgozat hátralévő része a következőképpen épül fel. A második fejezetben minőségbiztosításról lesz szó általánosságban, bemutatásra kerülnek a forráskódmetriák, verziókövető rendszer és a minősítő modell működése, felépítése. Harmadik fejezet az általunk létrehozott fejlesztői minősítő modellt mutatja be, annak elemeit és összefüggéseit. Negyedik fejezetben bemutatásra kerülnek a használt és az általunk fejlesztett rendszerek, valamint a fejlesztők kiértékelésének folyamata. Az ötödik fejezet az elért eredményeket mutatja be, röviden szó lesz az elemzett rendszerről, valamint a kiértékeléshez használt adatok begyűjtésének módjáról is. A hatodik fejezetben a kapcsolódó munkákról lesz szó. Végül a hetedik fejezetben összegezzük az elért eredményeket és pár lehetséges jövőbeli fejlesztést is megemlítünk.

2 Minősbiztosítás

Annak érdekében, hogy a fejlesztés alatt lévő szoftvert megfelelően lehessen irányítani, a menedzsereknek és a fejlesztőknek is ismerniük kell a szoftver tulajdonságait. A rendszer minőségbeli hiányosságainak és erősségeinek tudatában könnyebben becsülhetőek a fejlesztés további költségei, és előre fel lehet készülni az esetleges vészhelyzetek kezelésére. A következőkben a szoftver minőségének becslésére használható különböző technikákról lesz szó.

2.1 Forráskód metrikák

Szoftver metrikának nevezzük egy szoftver valamely tulajdonságának a mért értékét. Ezen mérések elengedhetetlenek a szoftver megfelelő menedzseléséhez. *„Nem tudod kezelni amit nem tudsz irányítani, és nem tudod irányítani amit nem mérsz.”* Tom DeMarco. Egy szoftver számos aspektusa mérhető, melyek főként forráskóddal vagy specifikációval kapcsolatosak.

Forráskód metrikák a szoftver implementációjának minőségének becslésére alkalmasak. A forráskód analizálásával különböző tulajdonságai mérhetők, melyek segítségével az egyes elemek összehasonlíthatóak vagy rossz előjelként foghatóak fel. Ezen metrikák öt fő csoportra [2] bonthatóak. Méret alapú metrikák, melyek a szoftver méreteiről adnak alapvető információkat. Mivel ezek egyszerű metrikák, így gyorsan számíthatóak és könnyedén megérthetőek. Öröklődés alapú metrikák, melyek a rendszer öröklődési fáját elemzik és a szoftver komplexitása becsülhető segítségükkel. Hasonlóan a szoftver bonyolultságának becslésére alkalmasak a komplexitás és kapcsolódás alapú metrikák, melyek a vezérlési szerkezetek számát és bonyolultságát, illetve az egyes forráskódelemek kapcsolatát mérik. A bonyolult kód nem kívánatos, mivel módosítások során nagyobb a hibalehetőség, illetve tesztelése is sokkal nehezebb, mivel nehezebb a teljes kódot lefedő teszteseteket készíteni. Az ötödik csoport a kohézió alapú metrikák, melyek az egyes kódelemek egységességét mérik. Minél nagyobb az egyes kódelemek egységbezárása, annál könnyebb azok felhasználhatósága más rendszerekben, ami a költségek csökkentésében játszhat nagy szerepet.

A fenti öt csoportba tartozó metrikákon kívül fontosak a forráskódban található kódelemek duplikációi (továbbiakban klónok), kódolási szabálysértések és rossz előjelek is. A klónok az egyes elemek lemásolásával keletkeznek, és a forráskód karbantarthatóságát

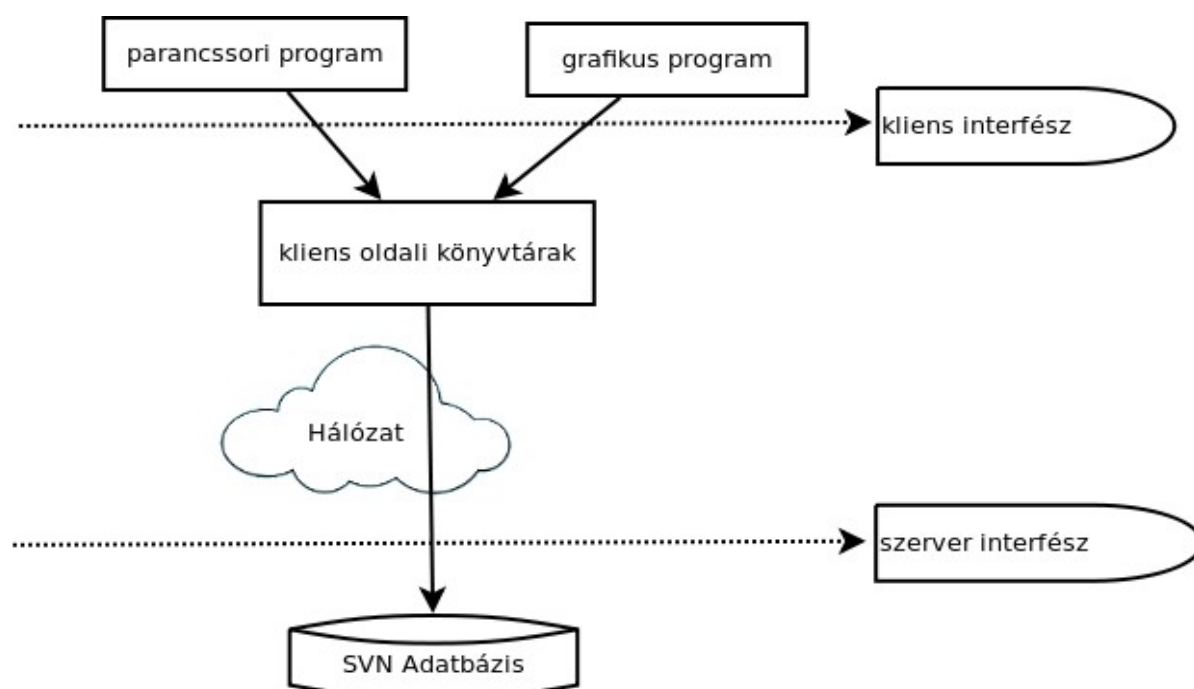
rontják. Abban az esetben, ha rossznak minősül egy ilyen kódrészlet vagy más okok miatt módosításra van szükség, minden klón példányon ugyanazt a változtatást el kellene végezni. Ez egyfelől többletmunkával jár, másfelől könnyedén megeshet hogy bizonyos példányok nem kerülnek javításra, így a régiek hibás állapotukban élnek tovább.

A kódolási szabálysértések [3] olyan hibák, melyek jelenléte mellett is fordítható és futtatható az adott rendszer, de speciális esetekben futási hibákhoz vezethetnek. Az ilyen típusú hibák elkerülése nagyon fontos, mivel sok esetben az általuk okozott hiba alapján nehéz felderíteni a rendellenesség igazi okát. Ilyen hiba az üres *catch* blokk (EmptyCatchBlock, PMD_ECB) használata egy kivétel kezelésekor. Ha az adott hiba dobásra kerül, akkor az üres *catch* blokk elnyeli azt, ahelyett hogy valamilyen módon értesítést küldene a felhasználó felé. Ezenkívül ily módon a program futása egy olyan állapotba kerülhet amelyben olyan hibák keletkezhetnek melyek alapján nem lehet következtetni az eredeti hibára. Egy kicsit elfogadhatóbb szabálysértés az üres *if* kifejezés (EmptyIfStmt, PMD_EIS) használata, melynek következménye hogy az adott feltétel vizsgálatra kerül, azonban a megfelelő utasítás blokk üressége miatt nem hajtódik végre semmi. Egy másik jelentős hiba kettő objektum összehasonlítása (CompareObjectsWithEquals, PMD_COWE). Nem mindegy, hogy az összehasonlítás alkalmával az `==` operátor vagy az `equals()` metódus van használva, mivel az első referencia szerint, a második pedig különböző tulajdonságok szerint hasonlítja össze a kettő objektumot.

A forráskódban a rossz előjelek [4] olyan mutatók, melyek arra utalnak, hogy valami talán helytelen lehet a kódban. Nem minden esetben mutatnak rá valódi hibára, azonban sokszor a kód valamilyen módosítására van szükség ahhoz hogy a hiba lehetőségét kizárjuk. A következőkben néhány rossz előjel kerül bemutatásra. Adat osztály (Data Class, DC) rossz előjel akkor fordul elő, ha egy osztály esetén az osztály lokális attribútumainak és az azokon valamilyen feladatot végrehajtó metódusok (az attribútumokat beállító és lekérdező metódusok nem számítanak) hányada nagyobb mint egy előre meghatározott érték. Hosszú paraméter lista (Long Parameter List, LPL) egy osztály metódusánál nemkívánatos, mivel nehezen értelmezhető és a használata is körülményesebb. Objektumorientált programozás esetén az osztályok lokális attribútumokat tartalmazhatnak, melyeket az egyes metódusok felhasználhatnak, így a metódusok csak a szükséges paramétereket kell hogy bekérjék.

2.2 Verziókezelő rendszerből kinyerhető információk

A verziókezelő rendszerekről általánosságban elmondható, hogy könyvtárak és fájlok kezelésére alkalmas olyan módon, hogy a rajtuk történő változásokat is eltárolja. Az időbeli változások a későbbiek során is megtekinthetőek lesznek. A hibás változtatások tehát nem eredményeznek esetleges adatvesztést, mert az úgynevezett verziózott adatok visszanyerhetőek. Az ilyen rendszereket a közvélemény sokszor „időgépként” tartja számon. Léteznek általános célú verziókövetők, melyek bármilyen típusú fájlt kezelnek. Ilyen a Subversion (SVN) is. Ezek az elterjedtebbek. Másik kategóriájuk speciális feladatot látnak el, vagyis csak néhány fájl típusra vonatkozóan működnek. Ezek sokszor egy bizonyos programozási nyelv fájljait jelentik. További jellemzőjük lehet, hogy egyedi eszközöket nyújtanak a szoftver fejlesztéséhez.



1. ábra: Az SVN vázlatos szerkezete

A nyílt forráskódú, ingyenes verziókezelő rendszerek közül az SVN az egyik legelterjedtebb. Egy központosított rendszer: egyetlen adatbázissal rendelkezik, mely tárolja a fájlok és könyvtárak különböző verzióit. A felhasználók különböző felületeken érhetik el ezt az adatbázist, akár hálózaton keresztül is. A tárolt adatok egyaránt kezelhetőek parancssori programmal és grafikus felülettel rendelkező programmal is. A rendszer vázlatos architektúrája látható az 1. ábrán.

A Subversion-féle revíziókban metaadatként el van tárolva többek között a feltöltés (commit) időpontja és a feltöltő (SVN-beli) azonosítója is. A megvalósított programunk erre a

tényre alapozik. Ez azt jelenti, hogy bármiféle módosítás egyértelműen egy felhasználóhoz tartozik. A forráskód minősége az azt fejlesztő személyektől (tekinthetők felhasználóknak) függ. Az alapötletünk szerint amennyiben a forráskódot tudjuk minősíteni, és az előzőekben említettek alapján a forráskódon végzett módosításokat felhasználóhoz tudjuk kapcsolni, akkor a felhasználó által végzett módosítások minősége alapján a fejlesztőt is tudjuk minősíteni. A módosítások beazonosíthatóak a log és diff funkciókkal. A funkciók neve az angol log és difference szavakból ered. A log tehát egyfajta naplózási adatokat jelenít meg. Ilyenek az említett metaadatok és egy további, számunkra fontos információ, a változtatott fájlok és könyvtárak relatív elérési útvonala. A diff funkció szövegfájlok összehasonlítására alkalmas. Megjeleníti a felhasználó számára a fájl két verziójának összehasonlításából eredő különbségeket. Általánosan ezek a különbségek lehetnek új sorok hozzáadása, sorok törlése vagy sorok módosítása.

2.3 Minősítő modell

A minősítő modell (quality model) [6] egy olyan körmentes irányított gráf, mely egy rendszer tulajdonságainak becslésére használható. Minden gráfpont a rendszer egy tulajdonságát reprezentálja, a pontok között lévő irányított élek pedig ezen tulajdonságok közötti függőségeket. Ezen szerkezet miatt a minősítő modellt attribútum függőségi gráfnak (Attribute Dependency Graph, ADG) is nevezhetjük. A gráfpontokat kettő csoportba sorolhatjuk aszerint, hogy értéküket milyen módon számítjuk. Azokat a pontokat, melyek nem rendelkeznek kimenő élekkel, belső pontoknak, a kimenő éllel rendelkező pontokat pedig a külső pontoknak nevezzük. A belső pontok a közvetlenül mérhető tulajdonságok alapján kaphatnak értékeket. Abban az esetben ha egy szoftver tulajdonságait szeretnénk elemezni, belső pontoknak tekinthetők a forráskód elemzésével kapott metrikákat reprezentáló pontok. Külső pontok absztraktabb tulajdonságoknak tekinthetők, nem közvetlenül mérhető tulajdonságok alapján kapnak értéket. Értékük más belső és külső pontok értékének bizonyos módon történő összesítésével számítható.

A modellben szereplő gráfpontok kiértékeléséhez szükségünk van egy referencia adatbázisra, melyben kiértékelhető elemek vannak. Ilyen elemek például a szoftver rendszerek (metrikái), vagy szoftverfejlesztők, pontosabban a róluk gyűjtött információk. Mivel az egyes tulajdonságokról nem tudni, hogy mi számít jónak illetve rossznak, a referencia adatbázis segítségével számítható a kívánt tulajdonságok átlagos értéke, mely

később egy konkrét elem értékelésekor összehasonlítási alapként szolgál. Vegyünk egy konkrét példát szoftverfejlesztők kiértékelésekor. Szeretnénk egy fejlesztőt az által minősíteni, hogy munkái során hány darab kódrészletet másolt le, azaz hány darab klónt hozott létre. Feltéhetjük, hogy referencia adatbázisunkban már szerepel több fejlesztő is, és ismerjük az általuk létrehozott klónok darabszámát. Ezen adatbázis segítségével kiszámolhatjuk az átlagosan létrehozott klónok számát, majd ebből az átlagos értékből kivonva az általunk vizsgálni kívánt fejlesztő klónjainak számát megkapjuk a vizsgált felhasználó átlaghoz viszonyított minősítését.

Minősítő modell használata esetén legtöbbször valamely absztraktabb tulajdonságra vagyunk kíváncsiak, azaz egy külső pont értékére. Annak érdekében hogy meghatározhassuk egy ilyen pont értékét, először a belső pontokat kell kiértékelni. A belső pontokat szenzoroknak is nevezhetjük, mivel az értéküket közvetlenül mérhető tulajdonságok alapján kapják. Minden szenzorra specifikálni kell azt a tulajdonságot, melyet a szenzor mér. Ezenkívül minden egyes tulajdonság esetében ismernünk kell azt is, hogy a mért értékek mit jelentenek. Pontosabban, hogy egy tulajdonság vizsgálatakor mi a mért érték jósága. Ezt súlyfüggvénnyel lehet érzékeltetni. Ha például azt szeretnénk kifejezni, hogy a nagyobb érték rosszabb teljesítményre utal, akkor a súlyfüggvény $f(x)=x$. A súlyfüggvény azért ez ebben az esetben, nem pedig $f(x)=-x$, mert egy forráskódhoz tartozó metrikaérték a legtöbb esetben annál jobb minél kisebb. Ha azt szeretnénk elérni, hogy a nulla érték a legrosszabb, az ettől eltérő értékek pedig jobbak akkor az $f(x)=-|x|$ függvényt kell súlyfüggvénynek választanunk. Ezen információk birtokában egy szenzor értékét a következőképpen számíthatjuk¹:

$$g(p) = \frac{\sum_{i=1}^{|T|} f(T_i)}{|T|} - f(T')$$

ahol p a számítani kívánt szenzor pont, T a referencia adatbázisban szereplő elemek p pont által meghatározott tulajdonság értékeinek vektora, T' pedig a vizsgálni kívánt elem megfelelő tulajdonságának az értéke. Tehát a referencia adatbázis alapján számított átlagos értékből vonjuk ki a vizsgált elem megfelelő értékét. Fontos megemlíteni, hogy egy a minősítő modellbeli pont értéke annál jobb minél nagyobb, ellenben a pont által mért konkrét tulajdonság (pl. metrika) értékével, melynek jóságát a fent említett súlyfüggvény határozza meg.

¹ Az említett kiszámítási mód egy egyszerűsített leírás, a pontos módszer a [6] referenciában olvasható.

A belső pontok ismeretében már kiszámíthatóak a modell külső pontjai is. A külső pontok más belső és külső pontok értékének aggregálásával számolható, ezért ezeket a pontokat aggregált pontoknak is nevezhetjük. Az aggregált pontok rendelkeznek a pontból kifelé vezető élekkel, ezek az élek határozzák meg a modell azon pontjait melyeket az adott pont értékének számításakor figyelembe kell venni. Minden él rendelkezik egy súllyal, azzal a feltétellel hogy egy aggregált pont kimenő éleinek össze-súlya 1 kell, hogy legyen. A súlyok azt határozzák meg, hogy az adott él cél pontja hány százalékban járul hozzá az aggregált pont értékéhez. Ezen kívül minden él rendelkezik egy pozitív vagy negatív előjellel is, mely azt határozza meg, hogy az adott él cél pontjának az értékét milyen értelemben kell felhasználni. Abban az esetben ha az előjel negatív, az adott gráfpont nagyobb értéke rontani (csökkenteni) fogja az aggregált pont értékét. Egy pont aggregált értékét a következőképpen számolhatjuk²:

$$h(q) = \sum_{i=1}^{n(q)} sign_i * \omega_i * g(p_i)$$

ahol q a számítandó pont, $n(q)$ a q pontból kimenő élek száma, $sign_i$, ω_i és p_i az i . kimenő élhez tartozó előjel, súly érték és cél pont. Ezzel azonban csak a q pont aggregált értékét számoltuk ki, a $g(q)$ érték meghatározásához a referencia adatbázisban szereplő elemekre számított átlagos $h(q)$ értékből ki kell vonna a vizsgált elemre kapott megfelelő értéket.

Egy adott elemre (például szoftverfejlesztőre) a teljes ADG kiértékelése után, már megmondhatjuk, hogy az egyes tulajdonságokban átlagon alul vagy felül teljesített-e, illetve ha több elemre is kiértékeljük a modellt, akkor az egyes elemeket össze is lehet hasonlítani az egyes tulajdonságok tükrében. Egy adott tulajdonság alapján az az elem a jobb, akinek a tulajdonsághoz tartozó modellbeli pontjának az értéke magasabb.

² Az említett kiszámítási mód egy egyszerűsített leírás, a pontos módszer a [6] referenciában olvasható.

3 Fejlesztői minősítő modell

Dolgozatunk egyik célja egy olyan minősítő modell megalkotása, mely segítségével az egyes szoftverfejlesztőket elvégzett munkájuk különböző vonatkozásaiban értékelni lehet. Az értékeléseket felhasználva meg tudjuk határozni, hogy az egyes fejlesztők az átlaghoz lépest, illetve egymáshoz képest milyen minőségű munkát végeztek. Ebben a fejezetben bemutatásra kerül az általunk szoftverfejlesztők minősítésére kifejlesztet minősítő modell, mely a 2 ábrán látható.

3.1 Szenzor pontok

A 2.3 fejezetben ismertetésre kerültek a szenzorokhoz tartozó súlyfüggvények. Említettük hogy az $f(x)=x$ súlyfüggvény jelentése az, hogy a nagyobb tulajdonság érték a rosszabbat jelenti. A szoftverfejlesztőkhöz tartozó legtöbb metrikát úgy alkottuk meg, hogy a nagyobb értékek jelentik a jobb teljesítményt, ezért súlyfüggvényeink negatív előjellel szerepelnek. Szenzorainknál kettő különböző súlyfüggvényt használtunk. Az egyik az $f(x)=-x$, a másik pedig az $f(x)=-\ln|x+1|$ függvény. Az első lineáris kapcsolatot fejez ki az egyes értékek között. A második esetben az egyes értékek között logaritmikus kapcsolat van (a függvényben azért növeljük az x változó értékét eggyel, hogy elkerüljük a nulla értéket a logaritmus függvényben.). Ez azt jelenti, hogy a nullához közelebbi x értékek közötti különbségek erőteljesebbek mint a nullától távolabb lévők. A következő szenzorokat hoztuk létre, melyekhez kapcsolódó metrikák angol neve és rövidítése zárójelen belül olvasható (ahol külön nem említjük a szenzorhoz tartozó metrika értékét számító eljárást, ott az értéket manuálisan kell megadni):

- **Dolgozott hónapok száma** (Number of Worked Months, DQ_NWM) – A személy által ledolgozott munkahónapok száma. Kizárólag szoftverfejlesztéssel kapcsolatos munka számítható be. Súlyfüggvény: $f(x)=-\ln|x+1|$, a logaritmikus súlyfüggvény oka az, hogy a fejlesztők munkájuk kezdeti szakaszában több újdonsággal találkozhatnak mint a későbbiekben, ezáltal több tapasztalatot szerezhetnek.
- **Dolgozott hónapok száma az aktuális helyen** (Number of Worked Months at Actual Workplace, DQ_NWMAWP) – Hasonló az előzőhöz, azzal a különbséggel hogy itt csak az aktuális munkahelyen eltöltött idő számít. Ez a szenzor azért létezik külön is,

mert az aktuális munkahelyen elvégzett munka nagy valószínűséggel hasonlít a jövőben elvégzendő munkákhoz, míg más munkahelyen lehet hogy más jellegű feladatokat kellett elvégeznie az adott személynek. Súlyfüggvény: $f(x) = -\ln|x+1|$

- **Ismert technológiák száma** (Number Of Known Technologies, DQ_NOKT) – Technológiának számítanak a különböző programozási nyelvek, keretrendszerek, fejlesztési és futtatási környezetek, szabványok, stb. Súlyfüggvény: $f(x) = -x$
- **Publikációk száma** (Number Of PUblications, DQ_NOPU) – Az adott személy által készített publikációk és egyéb tudományos dokumentumok száma. Súlyfüggvény: $f(x) = -\ln|x+1|$
- **Fizetési kategória** (Salary Category, DQ_SC) – Minden fejlesztőt egy kategóriába soroltunk be egytől tízig terjedő skálán fizetése alapján. Azt hogy melyik kategóriába tartozik egy személy, főként az általa betöltött munkakör és a végzettsége határozza meg. Súlyfüggvény: $f(x) = -x$
- **Végzettség** (Education Level, DQ_EL) – A következő végzettségi szinteket különítettük el: általános iskola, középiskola, felsőfokú alapképzés, felsőfokú mesterképzés, doktori fokozat. Minden fokozat egy számot határoz meg, az értékek a fokozatokkal növekednek. Súlyfüggvény: $f(x) = -\ln|x+1|$
- **Megbízhatóság** (Reliability, DQ_REL) – Megbízhatóság az alapján becsülhető, hogy az adott fejlesztőre milyen jellegű munkákat lehet bízni, figyelembe véve azok határidejét és bonyolultságát. Minden fejlesztőt egytől kilencig terjedő skálán soroltuk egy kategóriába, ahol a nagyobb értékű kategória a nagyobb megbízhatóságot jelenti. Súlyfüggvény: $f(x) = -x$
- **Kommunikációs készség** (COmmunicational Ability, DQ_COA) – A kommunikációs készség az alapján becsülhető, hogy az adott személy milyen mértékben tudja megértetni magát másokkal, illetve megérteni másokat. Hasonlóan az előzőhöz, minden fejlesztőt egytől kilencig terjedő skálán soroltuk egy kategóriába. Súlyfüggvény: $f(x) = -x$
- **Feladatok nehézsége** (TAsk Difficulty, DQ_TAD) – Az adott fejlesztő személyes véleménye arról, hogy a rá kiszabott feladatok számára mennyi nehézséget okoznak, mennyi új technológiával kell megismerkednie és milyen gördülékenyen tud haladni a feladattal. Az előzőekhez hasonlóan itt is egytől kilencig terjedő skálán soroltuk

kategóriákba az egyes személyeket, ebben az esetben a nagyobb érték a nehezebb feladatokat jelenti. Súlyfüggvény: $f(x) = -x$

- **Szakmai kihívás** (Technical Challenge, DQ_TCH) – Az érték az alapján határozható meg, hogy egy adott fejlesztőt mennyire motiválják a rá kiszabott feladatok. Például van olyan fejlesztő akit az motivál ha könnyen megoldható feladatokat kap, míg van olyan is akit az ha feladatai újszerűek és újat tanulhat elvégzésük során. Kilenc kategóriát határoztunk meg, ahol a nagyobb értékű a nagyobb kihívást jelenti. Súlyfüggvény: $f(x) = -x$
- **Munkakörnyezet minősége** (Environment Quality, DQ_EQ) – Az adott személy saját véleménye a környezetről, melyben dolgozik. Befolyásoló tényezők lehetnek az irodák felszereltsége, zsúfoltsága, munkaeszközök minősége, stb. Itt is kilenc kategóriába soroltuk a fejlesztőket. Súlyfüggvény: $f(x) = -x$
- **Projekt menedzser minősítése** (Qualification of Project Manager, DQ_QPM) – Az adott személy projektvezetőjének munkájával való elégedettsége határozza meg. Az előzőekhez hasonlóan kilenc kategóriát határoztunk meg. Súlyfüggvény: $f(x) = -x$
- **Létrehozott klónok száma** (Created Clone Smells, DQ_CCS) – Az aktuális szenzor az adott fejlesztő létrehozott klónjainak számát méri. A fejlesztő által létrehozott klónok számának meghatározási módja a következő. Meghatározzuk a vizsgált felhasználóhoz tartozó verziókat, azaz azokat a verziókövető rendszerbeli revíziókat, melyeket az adott fejlesztő hozott létre. Amennyiben ezekben a revíziókban vannak olyan klónok, amelyek az őket megelőzőben nem voltak, akkor ezeket a klónokat a vizsgált fejlesztő hozta létre. Súlyfüggvény: $f(x) = -x$
- **Létrehozott szabálysértések és rossz előjelek száma** (Created Rule Violations and Badsmells, DQ_CRVB) – A 2.1 fejezetben már ismertetésre kerültek olyan forráskódmetriák, melyek valamilyen szabálysértésnek vagy rossz előjelnek tudhatók be. Ez a szenzor az egyes felhasználók által kiváltott ilyen metrikák számát méri. Az adott felhasználóhoz tartozó ilyen típusú metrikákat a létrehozott klónok számításánál említett módszerhez hasonlóan lehet megállapítani, azzal a különbséggel, hogy itt nem a klónokat kell vizsgálni. Súlyfüggvény: $f(x) = -x$
- **Javított szabálysértések és rossz előjelek száma** (Fixed Rule Violations and Badsmells, DQ_FRVB) – Ez a szenzor az előzőnél említett metrikák javítását, azaz az

eltüntetett ilyen típusú metrikák számát méri. Az adott felhasználó által eltüntetett kódolási hibák számát az eddigi módszerhez hasonlóan végezzük. Meg kell határozni a fejlesztőhöz tartozó revíziókat, majd azon metrikák számát, melyek az egyel korábbi revízióban még léteztek, de az aktuálisban már nem. Súlyfüggvény: $f(x) = -x$

- **Határidők tartása** (Honour to Deadlines, DQ_HD) – Az érték az alapján határozható meg, hogy egy fejlesztő adott munkáinak határidejét mennyire tartotta a múltban, azaz hogy mennyire volt pontos. Kilenc kategóriába soroltuk a fejlesztőket, ahol a nagyobb értékű jelenti a nagyobb pontosságot. Súlyfüggvény: $f(x) = -x$

A következő négy darab szenzor azt méri, hogy egy szoftverfejlesztő heti munkaidejének hány százalékát töltötte adott tevékenységgel. A szenzorokhoz tartozó metrikák egy kivételével nullától százig vehetnek fel értéket, mely a százalékot fejezi ki.

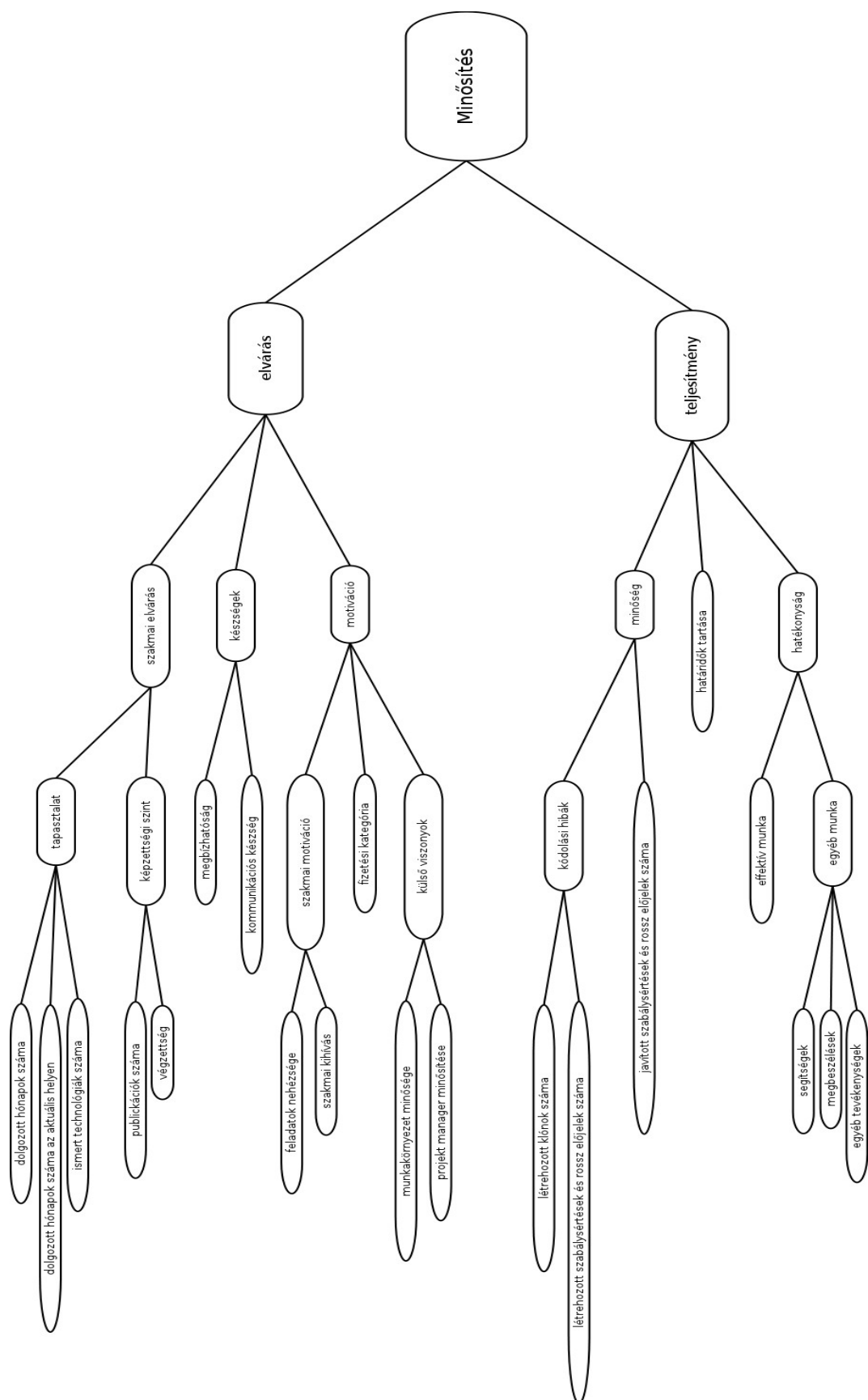
- **Effektív munka** (Time of Effective Work, DQ_TEW) – A szoftverfejlesztő fő munkakörére fordított idő hányada, azaz szoftverfejlesztés, annak dokumentálása, tesztelése. Súlyfüggvény: $f(x) = -x$
- **Segítség** (Time Spent by Assisting, DQ_TSA) – Ez a szenzor a fejlesztő segítségnyújtásra és segítségkérésre fordított munkaidejének százalékos különbségét méri. Számításhoz használt képlet a következő:

$$\left(\frac{\text{segítségnyújtás}}{\text{hetimunkaidő}} - \frac{\text{segítségkérés}}{\text{hetimunkaidő}} \right) * 100$$

A szenzorhoz tartozó metrika mínusz száztól százig vehet fel értéket. Súlyfüggvény:

$f(x) = -\ln|x + 101|$. (A függvényben azért növeljük az x változó értékét százeggyel, hogy elkerüljük a nulla értéket a logaritmus függvényben.)

- **Megbeszélések** (Time Spent by Meetings, DQ_TSM) – Különböző a fejlesztés alatt lévő projekttel kapcsolatos megbeszélésekre, tárgyalásokra fordított idő hányada. Súlyfüggvény: $f(x) = -x$
- **Egyéb tevékenységek** (Time Spent by Other Activity, DQ_TSOA) – Ez a szenzor az olyan tevékenységek hányadát méri, melyek a fenti három közül egyikbe sem sorolhatóak. Súlyfüggvény: $f(x) = -\ln|x + 1|$



2. ábra: Fejlesztői minősítő modell

3.2 Aggregált pontok

A minősítő modell aggregált pontja a modell más elemei által meghatározott értékeket összesíti, így egy absztraktabb tulajdonságot reprezentálnak. A következőkben ismertetjük az általunk létrehozott aggregált pontokat, azok kapcsolatait más modellbeli pontokkal. Az egyes gráfbeli élekre vonatkozó szavazatok itt nem kerülnek bemutatásra. A szavazatokról bővebb információ az 5.2 fejezetben található. Mivel a modellben szereplő legtöbb kapcsolat előjele pozitív, ezért ezt nem hangsúlyozzuk minden esetben, csak akkor ha az előjel negatív.

- **Tapasztalat** – Az egyes szoftverfejlesztők tapasztalatát fejezi ki, mely az adott fejlesztő eddigi munkái során megszerzett tudásra alapoz. Valószínűsíthető, hogy egy fejlesztő minél több munkahónappal rendelkezik, annál több tudást és gyakorlatot szerzett. Kapcsolódó pontok: dolgozott hónapok száma, dolgozott hónapok száma az aktuális helyen, ismert technológiák száma.
- **Képzettségi szint** – Az adott fejlesztő tudását fejezi ki, annak fényében, hogy milyen iskolai végzettséggel rendelkezik és mennyi a tudományos munkáinak száma. Kapcsolódó pontok: végzettség, publikációinak száma.
- **Szakmai elvárás** – Az adott fejlesztő tapasztalati és elméleti tudását reprezentálja, más szóval azt, hogy tudása alapján mekkora elvárásunk lehet az adott fejlesztőtől. Kapcsolódó pontok: tapasztalat, képzettségi szint.
- **Készségek** – Ez a pont a fejlesztők azon személyes tulajdonságait méri, melyek a munkájuk hatékony elvégzését befolyásolják. Ezek a tulajdonságok csak becsülhetők, az adott fejlesztővel kapcsolatban levő más fejlesztők és menedzserek által. Kapcsolódó pontok: megbízhatóság, kommunikációs készség.
- **Szakmai motiváció** – Egy adott fejlesztő feladataihoz való viszonyulását fejezi ki, azaz a megoldandó feladatok milyen mértékben okoznak nehézségeket, illetve mennyire inspirálóak. Kapcsolódó pontok: feladatok nehézsége, mely negatív előjellel számíthat, mivel a feladatok nehézségei csökkentik a fejlesztők motivációját; szakmai kihívás pozitív előjellel.
- **Külső viszonyok** – Ez a pont az egyes fejlesztők saját véleményére épít, azt illetően hogy mennyire vannak megelégedve környezetükkel. Környezetbe minden és mindenki beletartozik akivel az adott fejlesztő munkája során kapcsolatba kerül. Kapcsolódó pontok: munkakörnyezet minősége, projekt menedzser minősítése.

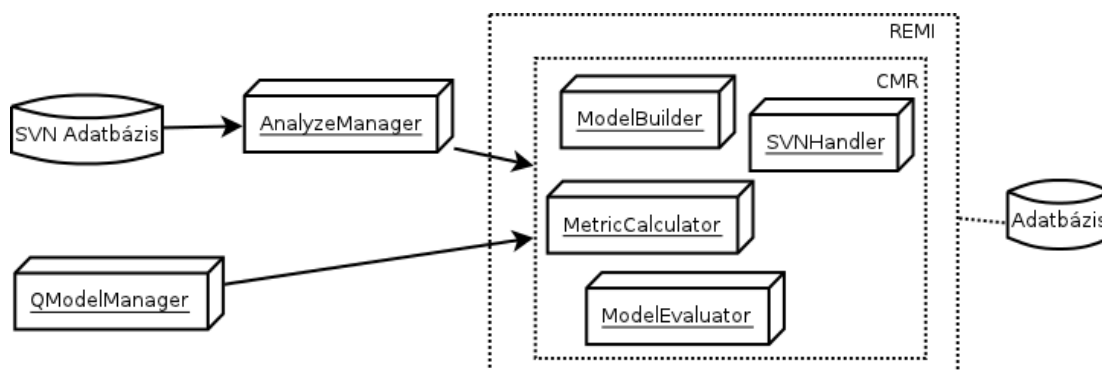
- **Motiváció** – A motiváció pont az olyan aspektusokat méri, melyek kihatással lehetnek az egyes fejlesztők munkamoráljára. Ez a pont is főként olyan információkat használ fel, melyek az egyes fejlesztők saját véleményei, ezért csak becsülni lehet. Kapcsolódó pontok: szakmai motiváció, fizetési kategória, külső viszonyok.
- **Elvárás** – Az elvárás a fejlesztői minősítő modell egyik főbb pontjának mondható. Az egyes fejlesztők tudása, képességei és véleményei alapján becsülhető segítségével egy adott fejlesztőtől elvárható munkaminőség. Kapcsolódó pontok: szakmai elvárás, készségek, motiváció.
- **Kódolási hibák** – Ez a pont a fejlesztők által létrehozott kódban szereplő hibák, szabálysértések számait elemzi. Fontos figyelembe venni, hogy legtöbb pont, mely a fejlesztői minősítő modellben szerepel, ha nagyobb értékkel rendelkezik, akkor az jobbnak számít. Ebben az esetben a nagyobb érték azt jelenti hogy a vizsgált fejlesztő az átlaghoz képest több hibát vétett programozás során. Tehát ennél a pontnál az alacsonyabb érték számít jobbnak. Kapcsolódó pontok: létrehozott klónok száma, létrehozott szabálysértések és rossz előjelek száma.
- **Minőség** – Hasonlóan az előző ponthoz, ez is az egyes fejlesztők által létrehozott kód minőségét méri, azonban ez a pont már egy szinttel feljebb található. A forráskód hibákon kívül a javított hibák számát is figyelembe veszi az értékek számításakor. Kapcsolódó pontok: kódolási hibák, mely negatív előjellel számíthatók a fent említett ok miatt, azaz mert a nagyobb kódolási hiba érték negatív irányban módosítja a létrehozott forráskód minőségét, a másik pont pedig a javított szabálysértések és rossz előjelek száma, mely pozitív előjellel számíthatók.
- **Egyéb munka** – Négy darab szenzort hoztunk létre a fejlesztők heti munkaidejének mérésére, vagyis arra, hogy az egyes fejlesztők hogyan osztják azt be, mivel töltik el. Ebből az egyik fontos szenzor a fejlesztésre szánt időt méri, azaz azt az időt, melyet kódolással töltött el. Az egyéb munka aggregált pont a nem fejlesztésre szánt munkákra szánt idő mérését végzi. Kapcsolódó pontok: segítségek, megbeszélések, egyéb tevékenységek.
- **Hatékonyság** – A hatékonyság pont a fejlesztők időbeosztásának mind a négy típusát vizsgálja közvetett módon. Kapcsolódó pontok: effektív munka, egyéb munka.
- **Teljesítmény** – A teljesítmény aggregált pont a fejlesztői minősítő modell másik

fontosabb eleme. Az elvárás aggregált ponttal ellentétben ez a pont az egyes fejlesztők teljesített munkájának minőségét méri. Pontosabban azt, hogy a munkát milyen minőségben és mekkora időtartam alatt végezte el. Kapcsolódó pontok: minőség, határidők tartása, hatékonyság.

- **Minősítés** – A kialakított minősítő modell legfelsőbb pontja a minősítés, mely minden eddigi tulajdonságot egybevetve minősíti az egyes fejlesztőket. A minősítés tehát egy érték, mely meghatározza hogy egy adott szoftverfejlesztő, a referencia adatbázisban lévő többi fejlesztőhöz képest milyen teljesítményt produkált. Kapcsolódó pontok az elvárás és teljesítmény. Tehát a minősítés felfogható úgy is mint az elvárás és a teljesítmény szembeállítása. Mivel az elvárás azt határozza meg, hogy mekkora teljesítményt várunk el az egyes fejlesztőktől, ezért ez a pont negatív előjellel számíthat az aggregálás során. Így ha egy fejlesztővel szemben magasak az elvárásaink, nagyobb teljesítményt kell nyújtania mint egy olyan fejlesztőnek akivel szemben alacsonyabbak az elvárásaink, annak érdekében, hogy a minősítések hasonlóak legyenek.

4 Megvalósítás

Az alábbi fejezetben ismertetjük az implementált rendszer architektúráját melyet a 3 ábra mutat be. Java nyelven készült el a Repository of Evolving Models and Instances (REMI) szerveralkalmazás Columbus Model Repository (CMR) nevű kiegészítéseként.



3. ábra: A megvalósított rendszer architektúráis szerkezete

A REMI egy adattárháznak tekinthető, mely egyedi funkcionalitással bír. Használatakor első lépésként egy modellt kell vele felépíteni és a későbbiekben csak ennek a modellnek megfelelő, korlátozásait, szabályait kielégítő adatokat vihetünk fel az adatbázisba. Fontos továbbá, hogy minden felvitt adat verziózva van. Így egy adat törlése vagy felülírása során ezen műveletek csak logikailag történnek meg, bármikor visszakereshetőek. Az adatmódosítás tehát két adat egymáshoz való viszonyának beállítását jelenti, mely egyik esetén az előző, másik esetén pedig a következő verziójának beállítása.

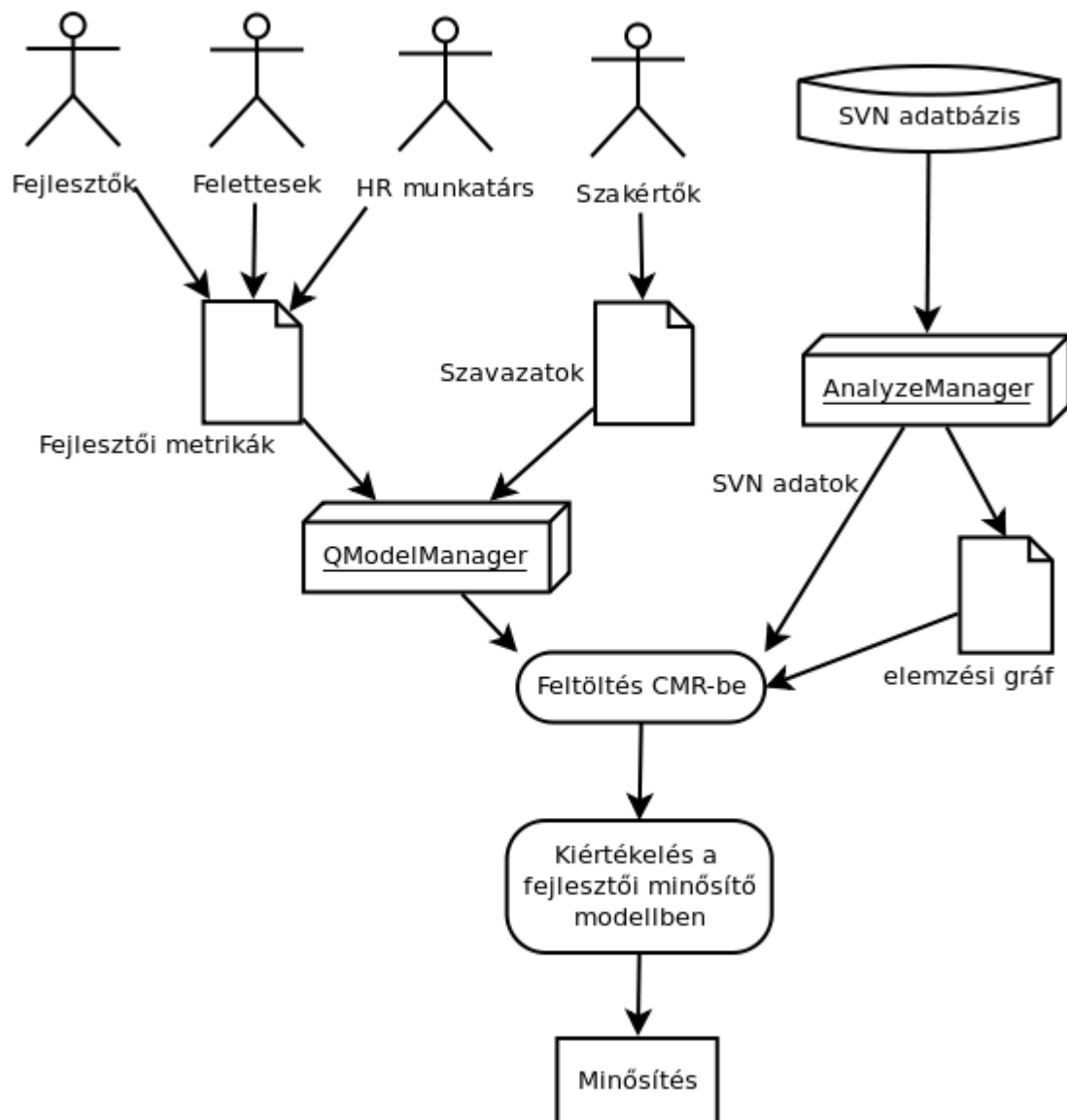
A REMI kiegészítését jelentő programok azt a célt szolgálják, hogy modellspecifikus funkcionalitást is tudjon a rendszer biztosítani szerveroldalon. Így a keretrendszer nem csak az általános felülettel rendelkezik a tárolt adatok hozzáférését, illetve felvitelét tekintően, sokkal kényelmesebb kezelhetőséget biztosítva ezzel. A CMR kiegészítés részeként valósítottuk meg munkánkat. Előző verzióiban már képes volt a rendszer statikusan elemzett forráskód szerkezetének és metrikáinak feltöltésére. Emiatt és a keretrendszer nyújtotta verziózási lehetőségek miatt célszerűnek tűnt ezt választani alapul.

A fejlesztői kiértékelési folyamat a szükséges adatok begyűjtésével és eltárolásával kezdődik. A Java forráskódelemzést az AnalyzeManager nevű eszközzel végeztük, melyet szükségleteink szerint továbbfejlesztettünk. Ez előállít egy gráf fájlt, melyet az említett CMR funkcionalitással eltárolunk az adattárházban. Minden ilyen elemzéshez tartoznia kell SVN adatok egy csoportjának, hiszen ez köti össze a forráskódot egy fejlesztővel. Ezen adatok

kinyerését a parancssori svn program automatizált használatával érjük el. CMR funkcionalitást biztosítottunk ezek eltárolására, melyet az SVNHandler modul végez. A 3.1 fejezetben leírt szenzor pontok mindegyikéhez társítani kell egy értéket minden fejlesztő esetén. Ezeket az értékeket a fejlesztőktől és feletteseiktől, illetve esetenként HR munkatárs segítségével gyűjtöttük be. A legtöbb kérdésre maga a fejlesztő is tudja a választ, néhány esetben a véleményére alapozunk, viszont vannak olyan szenzorok, melyek értékeit a felettesek jobban meg tudnak ítélni, vagy a HR munkatárs múltbéli adatok alapján pontos értéket tud szolgáltatni. Ezeket az adatokat egy-egy XML fájlban tároljuk fejlesztőnként. Azon szenzor pont értékeket, melyek automatikusan számíthatóak a fejlesztőhöz tartozó forráskód minősége alapján, szintén a CMR funkcionalitásaként implementáltuk (MetricCalculator modul). Az így összegyűjtött metrikaértékek feltöltésére is biztosítottunk felületet. Ezt a szolgáltatást a QModelManager nevű segédprogram veszi igénybe.

Amennyiben minden említett adat rendelkezésünkre áll és feltöltöttük őket a keretrendszerbe, megkezdődhet a kiértékelési folyamat. Fejlesztők referenciacsoportját kell megalkotni, melyekhez viszonyítva értékelhetünk egy fejlesztőt. Ez a referencia adatbázis. Szintén elkészítettünk egy interfészt ennek elvégzésére és a fejlesztői minősítő modell felépítésére, melyet a ModelBuilder modul végez. Ennek használata után minden rendelkezésre áll a folyamat végén szereplő CMR funkcionalitás használatához, az egyes fejlesztők kiértékeléséhez. Minden a minősítő modellekkel kapcsolatos műveletet, így ezeket is a QModelManager programmal végezzük, a CMR ModelEvaluator szolgáltatásán keresztül.

A teljes folyamatot tekintve az egyes lépések sorrendisége a következő. A fejlesztői metrikák, a minősítő modell szavazatainak begyűjtése és a forráskód minőségének elemzése történhet párhuzamosan. Következő lépésként az elemzési eredményeket tartalmazó gráfot töltjük fel a CMR rendszerbe. Ezt követi az SVN adatok feltöltése. Mindkettőt az AnalyzeManager program végzi. Ekkor a QmodelManager segítségével létrehozuk a fejlesztői minősítő modellt a CMR-ben az inputként kapott szakértői szavazatokkal. Ehhez a fejlesztők egy referencia csoportját is megalkotjuk, melyekhez viszonyítva tudunk később fejlesztőt minősíteni. Utolsó feltöltési lépésként a fejlesztőkhöz tartozó metrikákat töltjük fel, illetve számoljuk ki. Végso lépésként a QmodelManager programmal a megalkotott minősítő modellben elindítjuk a kiértékelést egy kiválasztott fejlesztőre. A folyamat a 4. ábrán tekinthető meg vizuálisan.



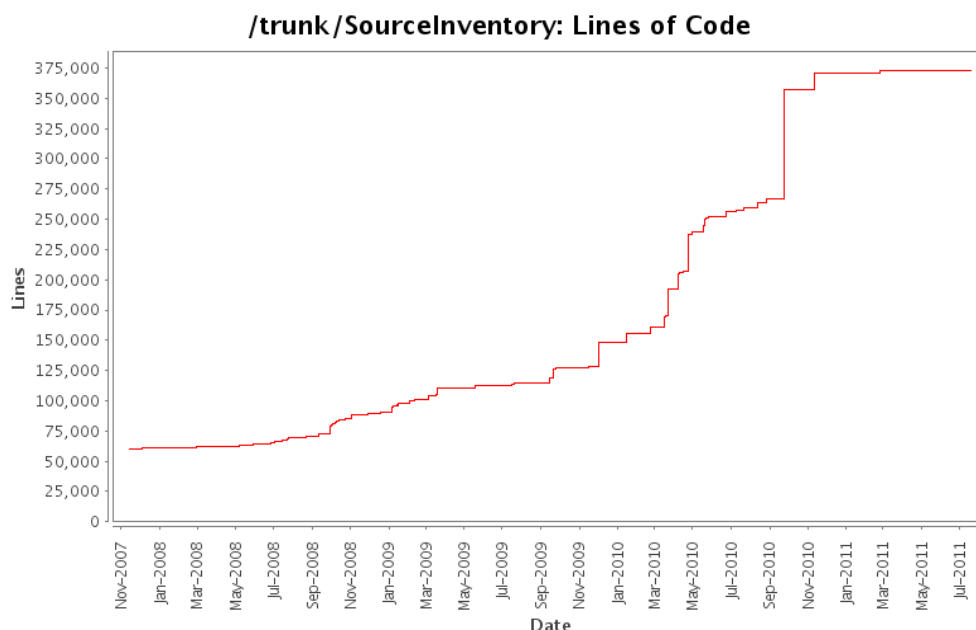
4. ábra: A kiértékelési folyamat

5 Eredmények

Az alábbi fejezetben prezentálásra kerülnek az elért eredmények. Röviden bemutatjuk az elemzett rendszert és azt, hogy az elemzéseket mely időintervallumon belül végeztük. Összefoglaljuk a fejlesztőkről begyűjtött adatokat, melyek egy részét maguktól a fejlesztőktől kértünk be, más részét azok vezetőitől, és egy kis hányadát a forráskódok illetve a verziókövető rendszer információi alapján számítottunk ki. Végül a fejlesztői minősítő modell által kiszámolt értékeket foglaljuk össze.

5.1 Elemzett rendszer

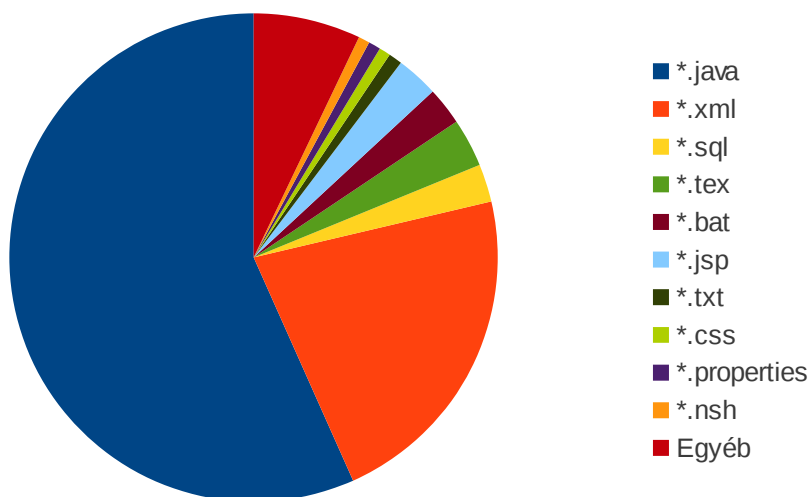
Tesztelés során a Szegedi Tudományegyetem és a FrontEndART Szoftver Kft. által közösen fejlesztett SourceInventory (SI) nevű rendszert elemeztük. Az SI egy olyan keretrendszer, mely más rendszerek fejlesztését segíti oly módon, hogy figyeli a létrehozott forráskódok minőségét. Így a menedzserek időben felderíthetik az esetleges klónokat vagy rossz előjeleket, és felszólíthatják a fejlesztőket azok javítására.



5. ábra: Source Inventory kódsorok számának változása

A szoftver java nyelven lett megvalósítva, illetve néhány segédalkalmazás esetében más technológiák is alkalmazásra kerültek, azonban mi az elemzések során csak a java forrásállományokat használtuk fel. Egy-egy revízió elemzésének időigényessége miatt nem

vettük figyelembe a verziókövető rendszerben levő összes revíziót mely az SI-vel kapcsolatos. Körülbelül ezerötyszáz revíziót elemeztünk ki a 4 fejezetben már említett AnalyzeManager nevű segédalkalmazással, mely revíziókat nyolc darab fejlesztő hozta létre. A teszteléshez használt referencia adatbázis ehhez a nyolc fejlesztőhöz kapcsolódó adatokból hoztuk létre.



6. ábra: SourceInventory fájltypusok eloszlása

Megjegyzés: a statisztikai adatokat a StatSVN [7] nevű alkalmazás szolgáltatta.

5.2 Gyűjtött adatok

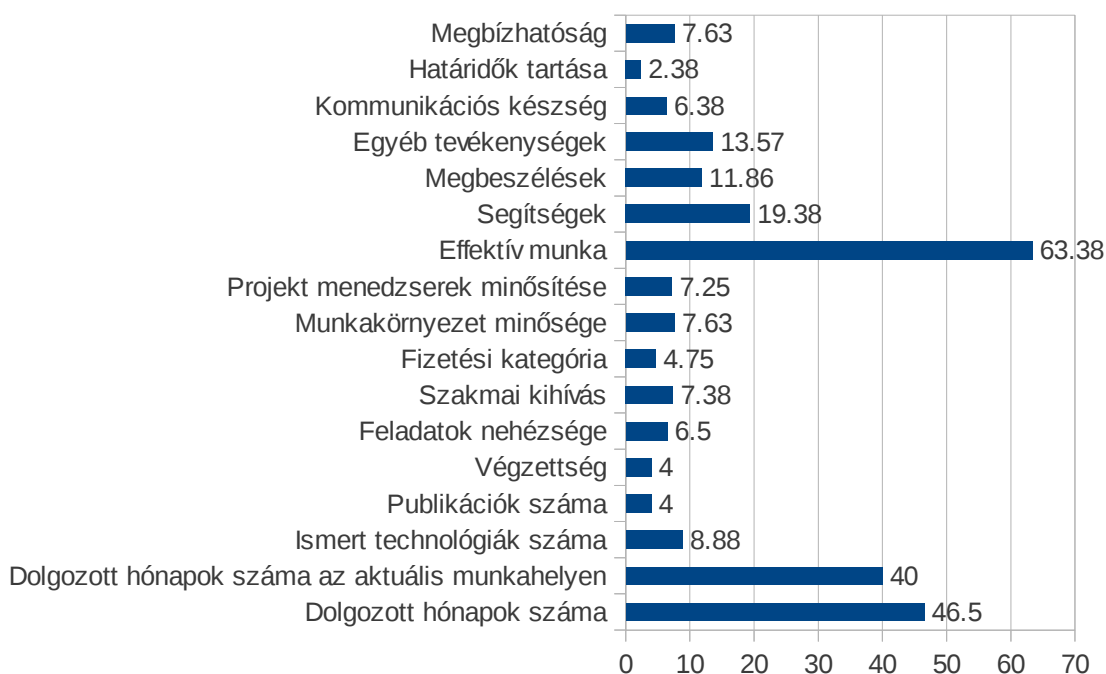
Egy adott minősítő modell működéséhez szükség van a szenzor pontok által mért tulajdonságok értékeire és az aggregált pontok kapcsolatainak súly értékeire. Ezeket az információkat különböző módszerek segítségével gyűjtöttük össze. Az adatok begyűjtése során számos személlyel vettük fel a kapcsolatot. A begyűjtött adatok átlagos értéke a 7 ábrán látható, az adatok értelmezéséhez a 3.1 fejezet használata ajánlott.

5.2.1 Fejlesztői adatok

A kidolgozott fejlesztői minősítő modell legtöbb szenzor pontja olyan tulajdonságokat értékel, melyek nem számíthatóak automatikusan az egyes fejlesztők által létrehozott forráskódok alapján. Ez miatt ezeket az adatok kérdőívek segítségével gyűjtöttük be a megfelelő személyektől. A kérdőíveket a Google Docs szolgáltatás segítségével alkottuk meg, melyek formája egy olyan weblap mely speciális beviteli mezőket tartalmaz, a különböző

céloknek megfelelően. A weblap által tartalmazott beviteli mezők kitöltése után egy gombbal van lehetőség a megadott információk elküldésére. Az információkat végül egy táblázatban kaptuk meg a Google Docs rendszer által. A begyűjtött adatok arra az időszakra vonatkoztak amikor az egyes fejlesztők az elemezett rendszeren dolgoztak. Tehát ha egy fejlesztő abban az időszakban amikor a Source Inventory rendszert fejlesztette, felsőfokú mester végzettséggel rendelkezett, de azóta már megszerezte a doktori fokozatát, akkor a felsőfokú végzettséget adta meg.

A minősítő modell által szükséges adatokat három módon gyűjtöttük be. Mivel az egyes fejlesztők fizetési kategóriái bizalmas adatok, így azt az ahhoz hozzáférő személytől kaptuk meg személyesen. A másik kettő kategóriába tartozó adatokat az online kérdőívek segítségével szereztük meg. Az egyik kategóriába tartoznak azok az adatok, melyeket az egyes fejlesztők saját magukról meg tudnak adni, mint például az, hogy hány hónapot dolgozott már, vagy az, hogy milyennek véli munkakörnyezetének minőségét. A másik kategóriába pedig azok az adatok tartoznak, melyeket az egyes fejlesztőkre azok vezetői tudnak megítélni, így ezeket a fejlesztők feletteseitől kértük be. Ebbe a kategóriába tartozik például a fejlesztő kommunikációs készsége.



7. ábra: Begyűjtött adatok átlagos értéke

Fontos megemlíteni, hogy néhány olyan tulajdonságot is értékel a minősítő modell, amelyek a fejlesztő saját véleményére építenek, és ezek szubjektivitása miatt az értékek a valóstól kicsit eltérőek lehetnek. Másik tényező ami az adatok pontosságát ronthatja az az, hogy a fejlesztőknek az értékeket nem a jelen állapotok szerint kellett megítélni, hanem pár évvel visszamenőleg.

5.2.2 Minősítő modell súlyai

A fejlesztői minősítő modell egyes aggregált pontok értékének számításához használt kapcsolatokra vonatkozó súlyok meghatározásában olyan személyek segítségét vettük igénybe, akik bizonyos mértékben ismerik a minősítő modell működését. A kifejlesztett minősítő modell ismertetése és átbeszélése után, ezektől a személyektől gyűjtöttük be a súly

Elvárás	0.35
Szakmai elvárás	0.33
Tapasztalat	0.66
Dolgozott hónapok száma	0.39
Dolgozott hónapok száma az aktuális helyen	0.28
Ismert technológiák száma	0.33
Képzettségi szint	0.34
Publikációk száma	0.38
Végzettség	0.63
Készségek	0.30
Megbízhatóság	0.54
Kommunikációs készség	0.46
Motiváció	0.38
Szakmai motiváció	0.43
Feladatok nehézsége	0.41
Szakmai kihívás	0.59
Fizetés	0.33
Külső viszonyok	0.24
Környezet minősége	0.47
Projekt menedzser minősítés	0.53
Teljesítmény	0.65
Minőség	0.37
Kódolási hibák	0.63
Létrehozott klónok száma	0.45
Létrehozott szabálysértések és rossz előjelek száma	0.55
Javított szabálysértések és rossz előjelek száma	0.37
Határidők tartása	0.30
Hatékonyság	0.33
Effektív munka	0.64
Egyéb munka	0.36
Segítségek	0.35
Megbeszélések	0.42
Egyéb tevékenység	0.23

1. Táblázat: Fejlesztői minősítő modell pontjaira adott súlyok átlaga

értékeket, melyeknek átlagos értékei az 1 táblázatban láthatóak. A táblázat a következőképpen értelmezhető: mivel az általunk megalkotott minősítő modellben minden pont csak egy darab másik pont leszármazottja (lényegében egy fa), ezért a táblázat soraiban az egyes kapcsolatok által meghatározott gyermek pontok találhatók, illetve az adott kapcsolat súlya. Megjegyzésként említenénk meg, hogy az egyes modellbeli pontok kimenő éleinek súlya egynek kell hogy legyen. Mivel az 1 táblázatban az átlagos értékeket tüntettük fel, így megeshet hogy kerekítésből adódóan az összsúly eltérő ettől.

5.3 Kiértékelés

A következőkben a fejlesztői minősítő modell segítségével az egyes fejlesztőkre kiszámított értékeket mutatjuk be különböző szemszögekből. Ahogy már említésre került, a verziókövető rendszer több revíziójához kapcsolódó adatok is elemzésre kerültek. Ezek a revíziók egy időbeliséget határoznak meg, melynek köszönhetően a minősítő modellt nem csak egy időpontban volt lehetőségünk futtatni, hanem különböző időpontokban is. A fejlesztői minősítő modell nagysága miatt nem tárgyaljuk egyszerre a modell összes pontjára kapott értékeket, hanem különböző részekre bontva a fontosabb aspektusokat emeljük ki.

5.3.1 Egy fejlesztő kiértékelése

A következőkben a minősítő modell lényegesebb pontjain végigmenve elemezzük az értékeket, melyeket a user3 fejlesztő kapott (2 táblázat) a kiértékelés során. Emlékeztetőül: a nulla közeli számok jelentik az átlagos értéket, a negatív illetve pozitív számok pedig az átlagon aluli, illetve átlagon felüli értékeket.

	user3	min	max
Minősítés	-0.9504	-6.5077	4.2038
Elvárás	-0.0768	-1.0039	0.3237
Szakmai elvárás	-0.4140	-1.1561	1.4591
Tapasztalat	-0.3328	-1.5709	2.0826
Dolgozott hónapok száma	-0.0375	-1.3625	0.7875
Dolgozott hónapok száma az aktuális helyen	0.0250	-1.8500	0.9625
Ismert technológiák száma	-0.8750	-4.8750	6.1250
Képzettségi szint	-0.4516	-0.4516	0.4906
Publikációk száma	-1.1250	-1.1250	1.4000
Végzettség	0.0000	0.0000	0.0000
Készségek	0.0844	-2.3031	1.8594
Megbízhatóság	0.3750	-1.6250	1.3750
Kommunikációs készség	-0.3750	-3.3750	2.6250
Motiváció	0.1866	-0.3737	0.1866
Szakmai motiváció	-0.2719	-0.8594	0.9031
Feladatok nehézsége	0.5000	-1.5000	0.5000
Szakmai kihívás	-0.3750	-1.3750	1.6250
Fizetés	-0.0875	-0.6000	0.6875
Külső viszonyok	1.5766	-1.0359	1.5766
Környezet minősége	1.3750	-1.6250	1.3750
Projekt menedzser minősítés	1.7500	-2.2500	1.7500
Teljesítmény	-1.4330	-10.1502	6.6520
Minőség	-7.6588	-14.9871	25.5411
Kódolási hibák	-8.6611	-18.7455	31.1982
Létrehozott klónok száma	0.6750	-0.3625	0.6750
Létrehozott szabálysértések és rossz előjelek száma	-16.2375	-33.7250	56.4250
Javított szabálysértések és rossz előjelek száma	-16.9375	-33.2250	56.8375
Határidők tartása	0.3750	-2.6250	1.3750
Hatékonyság	4.0426	-30.1754	19.1953
Effektív munka	6.6250	-48.3750	31.6250
Egyéb munka	-0.4198	-3.8589	4.1377
Segítségek	-0.0625	-0.2375	0.2125
Megbeszélések	-0.6875	-8.1875	9.3125
Egyéb tevékenység	-0.3875	-1.7750	1.2500

2. Táblázat: Fejlesztői minősítő modell értékei user3 fejlesztőre

A tapasztalat tulajdonsága a user3 fejlesztőnek az átlagosnál alacsonyabbnak mondható a kapott érték alapján, mely -0.3328 lett. Elemezve a tapasztalathoz kapcsolódó pontokat relevánsnak is tűnik. Dolgozott hónapok száma és a dolgozott hónapok száma az aktuális helyen pontok értékei közel vannak az átlaghoz, azonban az ismert technológiák száma már az átlagostól alacsonyabb.

A minőség pont az átlagnál jóval alacsonyabb -7.6588, de azért van olyan fejlesztő a referencia adatbázisban aki rosszabb értékkel rendelkezik. A minőség értéke az alábbiak szerint alakul ki. Először is a kódolási hibák pontot kell kiértékelni, mely alacsonyabb mint az

átlagos érték, -8.6611, azonban ennél a pontnál az alacsonyabb érték a kevesebb kódolási hibát jelenti. A kódolási hibák a létrehozott klónok és a létrehozott szabálysértések és rossz előjelek száma alapján számolódik. A user3 fejlesztő a létrehozott klónok számában a referencia adatbázis alapján a legjobb 0.6750, pontosabban a legrosszabb mert a legtöbb klónt hozta létre. Azonban ez a pont nem rendelkezik nagy differenciával ami a minimum és maximum értéket illeti. Így ez a tulajdonság nem növeli oly nagy mértékben a kódolási hibák pontot. Azonban a létrehozott szabálysértések és rossz előjelek pont minimum és maximum értékei között már nagyobb a különbség, valamint a fejlesztő -16.2375 értékkel rendelkezik, mely nagyobb hatással van a kódolási hibák pont értékének kialakulásában. A minőséget meghatározó másik pont a javított szabálysértések és rossz előjelek száma szenzor pont. Ebben a tulajdonságban is lemarad az adott fejlesztő az átlagtól, -16.9375 a kapott értéke. A minőség számításánál a kódolási hibák negatív előjellel szerepelnek, mivel minél több a kódolási hiba annál rosszabb a minőség. Fontos megemlíteni, hogy a -7.6588 nem a két kapcsolódó modellbeli pont átlaga, hanem a két pont szavazatokkal súlyozott átlagának a viszonyulása a referencia adatbázishoz.

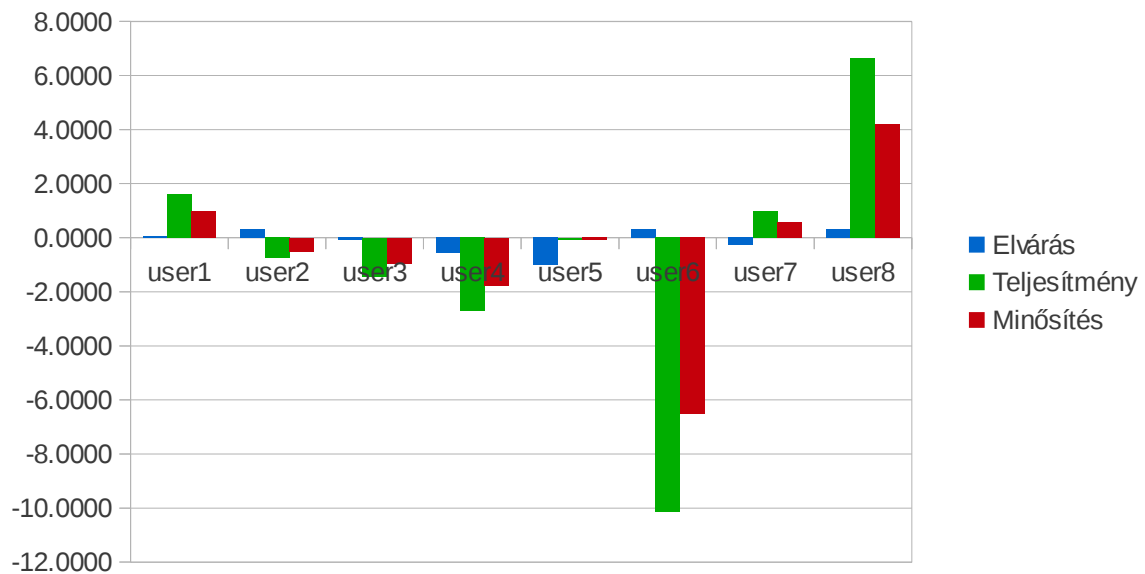
Minősítés a fejlesztői minősítő modell legfelsőbb pontja, mely az egyes fejlesztőket hasonlítja az átlaghoz a modell összes tulajdonsága alapján. User3 fejlesztő -0.9504 értéket kapott, ami az átlagnál alacsonyabb, viszont a minimum és maximum értékek (-6.5077, 4.2038) fényében átlagosnak mondható. A minősítés az elvárás (mely negatív értékkel számítható) és teljesítmény aggregált pontokból áll össze. A user3 fejlesztő esetében az elvárás -0.0768, mely az átlagnál alacsonyabb elvárást jelent, a teljesítmény pedig -1.4330 mely az átlagnál alacsonyabb, de közel átlagos teljesítményt jelent.

5.3.2 A modell főbb pontjai

Az alábbiakban a referencia adatbázisban szereplő összes szoftverfejlesztőre elvégeztük a minősítést egy adott időpont alapján. A kapott értékek a 3. táblázatban és a 8. ábrán láthatók.

	user1	user2	user3	user4	user5	user6	user7	user8
Elvárás	0.0541	0.2966	-0.0768	-0.5733	-1.0039	0.3237	-0.2602	0.2989
Teljesítmény	1.6070	-0.7395	-1.4330	-2.7064	-0.0698	-10.1502	0.9707	6.6520
Minősítés	0.9876	-0.5081	-0.9504	-1.7622	-0.0803	-6.5077	0.5818	4.2038

3. Táblázat: Fejlesztői minősítő modell három fő pontjának értékei



8. ábra: Fejlesztői minősítő modell három fő pontjának eloszlása

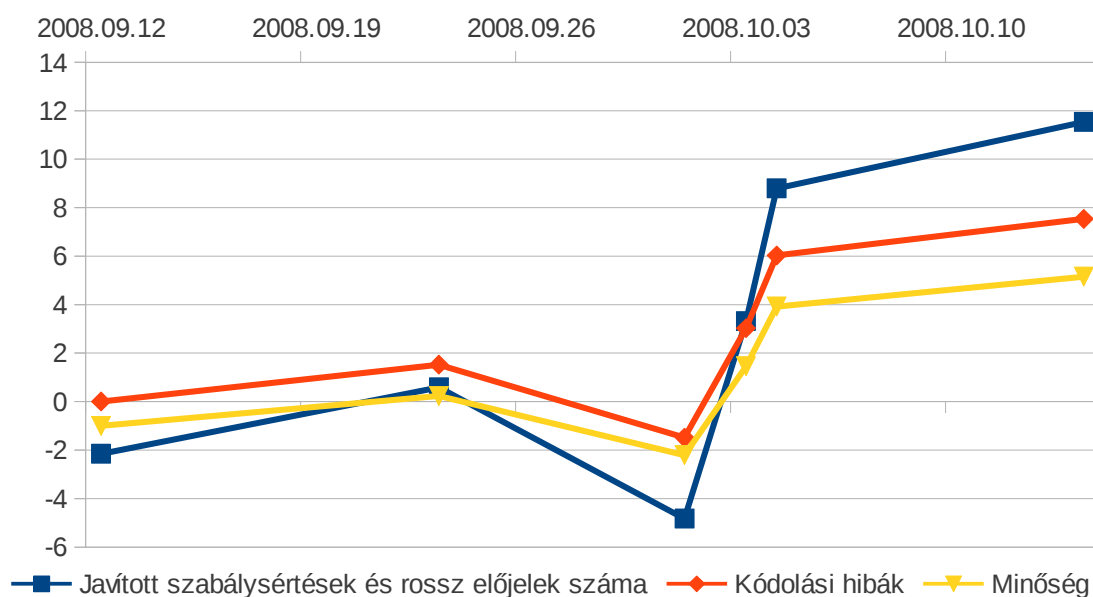
A fenti ábrán jól látható, hogy legjobb minősítést a user8 nevű fejlesztő kapta. Ennek oka az, hogy teljesítménye jóval nagyobb mint a referencia adatbázisban lévő többi fejlesztőé. A legrosszabb minősítést user6 kapta, ebben az esetben is a fő ok a teljesítmény, mely most sokkal alacsonyabb mint az átlag, ráadásul a hozzá tartozó elvárás a legnagyobb. Ezeken kívül látható hogy user5 nevű fejlesztőnél az elvárás alacsony, de a teljesítmény és a minőség közel átlagos. A közel átlagos minőség annak tudható be, hogy a szavazás során a szakértők a teljesítményt nagyobb súlyúnak vélték mint az elvárást.

5.3.3 Minőség alakulása az idő múlásával

Az alábbiakban a fejlesztői minősítő modell három pontjának értékének változását mutatjuk be az idő függvényében. Ez a három pont a Javított szabálysértések és rossz előjelek száma, kódolási hibák és az ezeket összefogó minőség. Mivel ezeket az értékeket meghatározó tulajdonságok a fejlesztés előrehaladtával folyamatosan változnak, így értelmet nyer az, hogy a fejlesztés különböző fázisaiban is elvégezzük az adott fejlesztők minősítését. Mivel az említett pontokhoz szükséges értékeket algoritmikusan számítani lehet a forráskód alapján, így nem kell minden időpillanatban a fejlesztőktől bekérni az adatokat. A 9 ábrán user3 fejlesztő által végzett kódolás minőségének alakulása látható kettő hónapnyi időszak alatt.

Az ábra alapján jól látható, hogy a mérések kezdetén a fejlesztő által létrehozott kódolási

hibák száma átlagosnak mondható. Ezzel szemben a javított szabálysértések és rossz előjelek pont értéke az átlagosnál alacsonyabb, aminek következményeként a minőség is alacsonyabb az átlagnál. A második mérési pontnál látható, hogy a javított hibák száma növekedett, sőt az átlagnál jobb lett, ami a minőséget is javította. Azonban mivel újabb kódolási hibák is keletkeztek, ezért a minőség nem növekedett oly mértékben mint a javított hibák pont. Következő mérési pontnál a javított hibák száma csökken az átlaghoz képest. Ennek oka az lehet, hogy az eltelt idő alatt a referencia adatbázisban lévő többi fejlesztő is módosította a forráskódot, és hibákat javítottak abban. Ennek következményeként user3 által javított hibák száma átlagon alulivá vált. Ötödik mérési pontnál látható, hogy a kódolási hibák és a javított hibák értéke is erőteljesen megnövekedett, melynek oka nagyobb mennyiségű kód létrehozása lehet az előző mérések óta. A minőség a következőképpen alakul. Mivel a javított hibák az átlagtól jóval magasabb így az a minőséget is nagy mértékben növeli, azonban a létrehozott kódolási hibák pont nem engedi hogy túl magas legyen, azaz az átlagos felé tolja a minőség értékét.



9. ábra: user3 által létrehozott forráskód minőségének alakulása az idő múlásával

5.3.4 Konklúzió

A fentiekben bemutatásra kerültek a fejlesztői minősítések különböző szempontok alapján. A fejlesztők feletteseivel folytatott megbeszélés során azt a következtetést vontuk le,

hogy a modell szerinti értékek meglehetősen tükrözték a valóságot. Láthattuk hogy a fejlesztők különböző tulajdonságok alapján összehasonlíthatóak, illetve az átlaghoz viszonyíthatóak. Azt is láthattuk, hogy az egyes elvontabb tulajdonságokat milyen tényezők befolyásolták. Ezek alapján kiszűrhetőek az egyes fejlesztőkre nézve azok a tulajdonságok, melyek teljesítményüket rontják, és javíthatunk azokon. Például ha kiderül, hogy egy adott fejlesztő rossz minőségű kódot hoz létre, akkor különböző tanfolyamok keretében fejleszthetjük kódolási képességeit. Vagy ha azt kapjuk, hogy motivációja alacsony, meg kell határozni azokat a tényezőket melyek ezt okozzák, és a megfelelő módon javítani kell rajta annak érdekében, hogy a fejlesztő továbbra is az adott cégnél maradjon.

6 Kapcsolódó munkák

A fejlesztők minősítése egy kényes kérdés. Az irodalomban nincs sokat feszegetve a téma. A közvélemény szerint illetlen az ilyen célú törekvés, mert feltételezhető, hogy mindenki legjobb tudása szerint fejleszt. Sokan pedig nem lennének kíváncsiak az eredményekre az elkeseredettség megelőzése végett. A legtöbb kapcsolódó cikk kizárólag a programozói produktivitást méri, javaslatokat tesznek ennek mérésére, növelésére.

Tóth Gabriella és társai egy Eclipse fejlesztői környezet Productivity Plugin nevű kiegészítővel gyűjtöttek úgynevezett mikro-produktivitási adatokat a fejlesztők tevékenységeiről, így szándékoztak előrejelzést adni különböző típusú jövőbeni módosítások erőforrásigényére [8]. Szintén ilyen értékekre alapozva megmutatták, hogy a termék- és folyamatmetrikák együttes használatával alkotott modellek sokkal hatékonyabbak a komplexitás alakulásának és hibák előrejelzésének esetében mint a csak termékmetrikákon alapuló modellek esetében [9]. Atkins D.L és társai egy bizonyos fejlesztői eszközről mutatták meg, hogy a produktivitást akár 40%-kal is képes javítani [11].

A folyamatmetrikák régóta kutatott témakör. Khoshgoftaar társaival egy munkájában [10] szoftvermodulok megbízhatóságára ad becslést ezek használatával. Megerősíti azt az intuíciót, miszerint gyakrabban fordul elő hiba olyan kódrészletekben amelyeket változtatni kell előre nem látható okokból.

Laila Cheikhi társaival egy tanulmányában összeveti az ISO és IEEE produktivitásra vonatkozó szabványait [12]. Rámutat együtt való alkalmazhatóságukra. Ugyanakkor hiányosságokat is talál, de az eltérő lehetséges igények miatt (pl. funkciópontok számítása, dokumentáció fontossága) nagyon nehéz feladat lenne egy általános modellt megalkotni.

Az gyakorlatban a produktivitás mérésére legelőször a kódolt sorok számát használták fel. Az átlagos érték függött a választott programozási nyelvtől. Hamar kiderült, hogy ez nem hatékony módszer. Megalkották a funkciópont alapú számítást, melyben minden egyes funkció elkészítéséhez egy értéket rendeltek [16]. Az időegység alatt elkészített funkciókhoz tartozó értékek összege jelentette egy fejlesztő produktivitását. Az értékeket nehéz helyesen megítélni. A módszer ugyan közel 30 éves, de még mindig van törekvés naprakészen tartására [15].

Egy projekt sikerességét megítélően nem elhanyagolható tényező a használt folyamat. A folyamatok fejlesztésére alakult meg a Capability Maturity Model (CMM) [17]. Ebből

keletkezett később a Software CMM, mely szoftverfejlesztési projekt esetére lett kifejlesztve, a Capability Maturity Model Integration (CMMI), mely általános célú, valamint a People Capability Maturity Model, ami elsősorban a munkások jólétére, emberi erőforrási kérdésekre vonatkozóan ír le folyamatot [18].

7 Összefoglaló

Dolgozatunkban bemutattunk egy kísérleti módszert fejlesztők minősítésére és alkalmaztuk egy ipari projektre. A szükséges input adatokat a fejlesztőktől, feletteseiktől gyűjtöttük be, valamint a vizsgált projekt forráskódjából és a használt verziókezelő rendszer adatbázisából állítottuk elő. A fejlesztők kiértékelését a minősítő modell segítségével végeztük el. Az így kapott értékeket (természetesen név nélkül) ismertettük. A minősítő modell ismeretében intuitív magyarázatot adtunk néhány értékre, levezetve, hogy hogyan számolódott az ki.

Úgy gondoljuk, hogy a fejlesztők minősítésének témaköre a tudományban nem került még előtérbe, pedig igény van rá. Ily módon is lehetne optimalizálni egy projekt költségét, legyen az már aktív vagy jövőbeli. A forráskód minőség elemzés és fejlesztői produktivitás mérés nem elégíti ki ezt az igényt teljesen.

Kísérletünket számos úton lehet folytatni. Az input adataink néhány esetben (ilyen pl. a vizsgált fejlesztők végzettsége) túlságosan homogén volt. Olyan projekten is kellene elemzéseket végezni, ahol minden adat sokszínű, vagy legalább minél kevesebb homogén adat van. Tekintve, hogy begyűjtött adataink múltbéliiek, azok változására csak nagyon kevés esetben tudtunk figyelni. Szeretnénk egy jövőbeli projekten is kísérletet végezni ahol bizonyos időközönként beszerzünk minden input adatot. Ez lehetőséget adna arra, hogy javítani tudjunk az fejlesztők gyengeségein. Néhány ilyen javítási próbálkozás lehetne például tanfolyamra való elküldése amennyiben az átlagostól több implementációs hibát vét, felhívni figyelmét a rossz időbeosztásra, valamilyen módon változtatni munkahelyi környezetén ha ez okozza rossz minősítését, de akár az is látszódná a modellből, hogy a projektvezető nem kívánt személy a fejlesztők körében és fontolóra kellene venni lecserélését. Szeretnénk az előbbieket alkalmazása után újra megvizsgálni ugyanazon fejlesztőcsapat minőségét és megállapítani, hogy hatásos volt-e a javítási törekvés.

8 Irodalomjegyzék

- [1] ISO/IEC 29119 Software Testing Standard <http://softwaretestingstandard.org>
- [2] Object-oriented metrics for Java, Rule Package Description for SourceAudit <http://www.frontendart.com/sites/default/files/resources/MET-java.pdf>
- [3] PMD programming guidelines for Java, Rule Package Description for SourceAudit <http://www.frontendart.com/sites/default/files/resources/PMD.pdf>
- [4] Bad smell miner, Rule Package Description for SourceAudit <http://www.frontendart.com/sites/default/files/BSM.pdf>
- [5] Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato: *Version Control with Subversion*, O'Reilly Media; 1 edition (June 29, 2004), ISBN-10: 0596004486, ISBN-13: 978-0596004484
- [6] Tibor Bakota, Péter Hegedűs, Péter Körtvélyesi, Rudolf Ferenc, Tibor Gyimóthy : *A Probabilistic Software Quality Model*, 27th IEEE International Conference on Software Maintenance (ICSM), pages 243-252, 25-30 Sept. 2011.
- [7] StatSVN <http://statsvn.org/>
- [8] Tóth, G., Végh, Á. Z., Beszédes, Á., Schrettner, L., Gergely, T. and Gyimóthy, T. *Adjusting Effort Estimation Using Micro-Productivity Profiles*. In Proceedings of the 12th Symposium on Programming Languages and Software Tools (SPLST 2011), pages 207-218. Tallin, Estonia, October 5-7, 2011.
- [9] Gabriella Tóth, Ádám Zoltán Végh, Árpád Beszédes and Tibor Gyimóthy: *Adding Process Metrics to Enhance Modification Complexity Prediction*. 2011 IEEE 19th International Conference on Program Comprehension (ICPC), pages 201 – 204, Kingston, Ontario, Canada June 22-24, 2011.
- [10] Khoshgoftaar T. M., Allen E.B., Halstead R., Trio G.P., Flass R.M.: *Using process history to predict software quality*. Computer, Volume: 31, Issue: 4, pages 66 – 72, April 1998.
- [11] David L. Atkins, Thomas Ball, Todd L. Graves and Audris Mockus : *Using version control data to evaluate the impact of software tools: a case study of the Version Editor*. IEEE Transactions on Software Engineering, Volume: 28, Issue: 7, pages 625 – 637, July 2002.

- [12] Laila Cheikhi , Rafa E. Al-Qutaish , Ali Idri : *Software Productivity: Harmonization in ISO/IEEE Software Engineering Standards* . Journal of Software, vol. 7, no. 2, February 2012 , pages 462 – 470.
- [13] ISO, ISO/IEC TR 9126-4: Information Technology - Product Quality - Part 4: Quality in Use Metrics, Geneva, Switzerland: International Organization for Standardization, 2004.
- [14] IEEE, Std 1045: IEEE Standard for Software Productivity Metrics, New York, USA: Institute of Electrical and Electronics Engineers, 1992.
- [15] Wei Xia a, Luiz Fernando Capretz , Danny Ho c, Faheem Ahmed : *A new calibration for Function Point complexity weights* . Information and Software Technology Volume 50, Issues 7–8, June 2008, pages 670 – 683.
- [16] Function Point Counting Practices Manual, 4. kiadás (2004), International Function Point Users Group.
- [17] Mark C. Paulk : *A History of the Capability Maturity Model for Software* . The Software Quality Profile vol. 1, issue 1 , pages 5 – 19, 2009.
- [18] Bill Curtis , Bill Hefley , Sally Miller : *People Capability Maturity Model , (P-CMM) Version 2.0, Second Edition* , July 2009.
- [19] Boehm, B. W., C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. J. Madachy, D. Reifer, and B. Steece. 2000. *Software cost estimation with COCOMO II*. Upper Saddle River, N.J.: Prentice Hall.

9 Köszönetnyilvánítás

Ezúton szeretnénk köszönetet nyilvánítani témavezetőnknek Dr. Ferenc Rudolfnak és külső konzulensünknek Bakota Tibornak a dolgozat készítéséhez nyújtott segítségükért, valamint Siket Péternek aki az implementációban volt nagy segítségünkre. Ezen felül szeretnénk megköszönni mindazok segítségét, akik a szükséges adatokkal hozzájárultak munkánk elvégzéséhez.