

Miskolci Egyetem

Gépészmérnöki és Informatikai Kar

Általános Informatikai Intézeti Tanszék



Automatizált teljesítményértékelés a programozásban

TDK-DOLGOZAT

Készítette:

Fekete Antal Kristóf

B5T3ZN

mérnökinformatikus

MSc hallgató, alkalmazásfejlesztő szakirány

II. évfolyam

Témavezető:

Dr. Baksáné Dr. Varga Erika

egyetemi docens

Miskolc, 2022. november

Eredetiség nyilatkozat

Alulírott Fekete Antal Kristóf, a Miskolci Egyetem Gépészmérnöki és Informatikai Karának hallgatója büntetőjogi és fegyelmi felelősségem tudatában kijelentem és aláírással igazolom, hogy ezt a dolgozatot saját magam készítettem, a benne leírt vizsgálatokat – ha ezt külön nem jelzem – magam végeztem el, és az ismertetett eredményeket magam értem el. Adatokat, információkat csak az irodalomjegyzékben felsorolt forrásokból használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Miskolc, 2022. november 8.

.....

a hallgató aláírása

A konzulens nyilatkozata

Alulírott Dr. Baksáné Dr. Varga Erika a Miskolci Egyetem Informatikai Intézetének oktatója az **Automatizált teljesítményértékelés a programozásban** című TDK dolgozatot beadásra alkalmasnak ítélem.

Miskolc, 2022. november 8.



a konzulens aláírása

Tartalomjegyzék

1. Bevezetés	1
2. Irodalomkutatás, RPA	2
2.1. RPA előnyei az üzleti életben [5]	2
2.2. Alkalmazási területek	3
2.3. RPA alkalmazása a szoftverfejlesztés munkafázisaiban.....	5
3. A szoftvertesztelés és a programozói teljesítménymérés mutatói [15]	9
3.1. Megbízhatóság	9
3.2. Karbantarthatóság	10
3.3. Tesztelhetőség.....	11
3.4. Hordozhatóság	11
3.5. Újrafelhasználhatóság.....	12
4. A GitHub alkalmazása RPA eszközként	13
4.1. Hallgatói csoport és a kurzus jellemzése	13
4.2. Előkészületek	13
4.3. Alkalmazott technológiák és szolgáltatások	14
4.4. Megvalósítás	15
4.5. Működés a gyakorlatban	21
4.5.1. Bemutatás oktatói nézőpontból	21
4.5.2. Bemutatás hallgatói szemszögből	22
5. Konklúzió	26
6. Conclusion	27
7. Irodalomjegyzék	28

1. Bevezetés

A különböző folyamatok automatizálása iránt egyre jelentősebb igény mutatkozik. Ezek a rendszerek akár a nap 24 órájában képesek dolgozni kevesebb hibalehetőséggel, bármínemű kifáradás vagy betegség lehetőségének felmerülése nélkül, ezért alkalmasak lehetnek a munkaerő-piaci problémák javítására, költségcsökkentésre, valamint a hatékonyság növelésére is. Az automatizálásra való törekvés ugyanakkor nem a közelmúltban jelentkezett először. Elegendő például a forgalomirányító közlekedési jelzőlámpákat tekinteni, amelyek már hosszú ideje előre programozottan automatikusan szabályozzák a közúti forgalmat, így irányításukhoz nem szükséges élőerős munkaerő.

A COVID-19 veszélyhelyzet idején a távolléti oktatás új módszerek és eszközök alkalmazását tette szükségessé. Az informatikus hallgatók programozás oktatásában ezt megelőzően is igénybe vettek különböző szoftvereket és információs rendszereket, de ebben az időszakban kifejezetten fókuszba kerültek a távoktatást támogató alkalmazások. Bár azóta visszatértünk a jelenléti oktatásra, az új módszerek közül érdemes továbbra is használni azokat, amelyek az oktatás eredményességét és az oktatói munka hatékonyságát növelik.

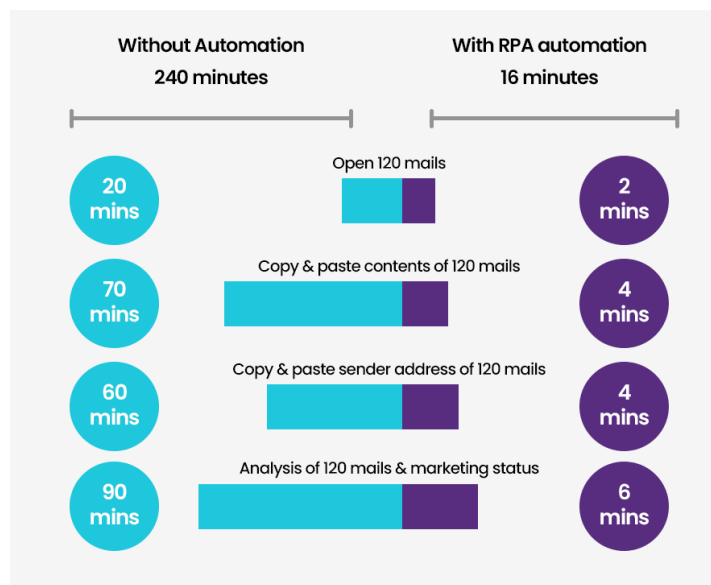
Az ipar 4.0 koncepció [1] a gyártási folyamatok legújabb informatikai szabványok alkalmazásával történő további stabilizálása mellett a szoftvertermékek előállításával, karbantartásával és terjesztésével, valamint a hozzájuk kapcsolódó szolgáltatások biztosításával foglalkozó szoftveripar területén is jelentős változásokat idézett elő. A fejlesztés során egyre többféle és gazdagabb funkció-készlettel rendelkező segédeszközt használnak. Az egyik igen elterjedt felhő-alapú szolgáltatás, ami a szoftverfejlesztők munkájának megkönnyítését és a hatékony csapatmunkát támogatja, a GitHub. Ennek célja, hogy könnyen kezelhető webes felületet biztosítson a programkódok tárolásához, verziókövetéséhez és megosztásához. Másrészt a gyakran ismétlődő, emberi döntést nem igénylő munkafolyamatok automatizálása (Robotic Process Automation, RPA) ebben az iparágban is fontos hatékonyságnövelő tényező. A szoftverfejlesztés során tipikusan automatizálható munkafázis például a szoftverek tesztelése, vagy a programozók teljesítményének a mérése. Jelen kutatás célja azt megvizsgálni, hogyan állítható a GitHub az RPA szolgálatába. Annak érdekében, hogy a jövő szoftverfejlesztői sikerrel tudjanak beilleszkedni az újfajta ipari környezetbe, már az informatikus alapképzési szakokon be kell vezetni az automatikus kód kiértékelő eszközök használatát. Jelen tanulmányban ezt az oktatásban rendelkezésre álló kapacitások hatékony kihasználása szempontjából vizsgálom.

2. Irodalomkutatás, RPA

A robotizált folyamatautomatizálás (Robotic Process Automation, RPA) [2] olyan eszközök gyűjtőfogalma, amelyek más számítógépes rendszerek felhasználói interfészén úgy működnek, ahogyan egy ember tenné. Olyan technológia, amelynek célja az emberi beavatkozás csökkentése a számítógépes alkalmazásokban. Különösen használatos ismétlődő feladatok esetén, ahol az egyes ismétlések között nagyon minimális a változás. Az RPA elsősorban magasszintű, többnyire a grafikus interfész szintjén lévő alkalmazásokkal való interakció révén működik, szemben a gépi nyelvvel vagy a programkóddal. Tehát egy olyan szoftvertípus, amely azt a valódi interakciót emulálja, amelyet egy ember a hagyományos számítógépes alkalmazásokkal folytatna [3]. Az üzleti folyamatok vonatkozásában az RPA kifejezés leggyakrabban a szoftverek konfigurálására utal, hogy automatikusan elvégezzék a korábban emberek által végzett tevékenységeket. Ilyen feladatok például az adatok átvitele különböző bemeneti (input) forrásból vállalatirányítási rendszerekbe (ERP, CRM stb.) [4]. Ebből következően az alkalmazottaknak több idejük marad arra, hogy a vállalat számára más, jelentősebb értéket képviselő tevékenységekre, például a döntéshozatalra vagy az ügyfélkapcsolat javítására összpontosítsanak. Viszonylag gyorsan bevezethető technológia, ezért azonnali előnyökkel járhat a vállalat számára idő- és költségmegtakarítás révén, különösen amennyiben bizonyos folyamatok szűk keresztmetszeteiben is alkalmazható [3].

2.1. RPA előnyei az üzleti életben [5]

1. Termelékenység növelése: a legtöbb RPA robotot úgy tervezték, hogy meghatározott rutinfeladatok elvégzésére összpontosítson. Például amennyiben egy alkalmazottra több órányi feladatterhet ró egy jelentés elkészítése, az RPA lehetővé teszi ezen időigény töredékére való csökkentését. Tehát az RPA segítségével ugyanaz az eredmény rövidebb idő alatt elérhető.
2. Hatékonyság növelése: az emberi hatékonyság korlátozott, mivel egy munkavállaló egy nap alatt csak adott mennyiségű időt tud a feladatra rászánni. Ez az időigény függ a dolgozó egyéni képességeitől és a kapacitásától is. Az RPA szoftvereknek ugyanakkor nincs szüksége szünetre, szabadságra vagy pihenőidőre, és attól sem kell tartani, hogy esetleg betegállományba kerülne.



1. ábra. RPA hatékonysága e-mail kezelés esetén [5]

A fenti 1. ábrán látható, hogy RPA automatizációval jelentősen lerövidülnek az egyszerű munkafolyamatok. A példában 120 db e-mail megnyitásához, a 120 db e-mail tartalmának, a küldő elektronikus levélcímének másolásához és beillesztéséhez, majd a levelek elemzéséhez szükséges idő került összehasonlításra. A világosabb színnel jelölt bal oldalon automatizálás nélkül, míg a sötétebb színnel ábrázolt jobb oldalon RPA automatizálás mellett került jelölésre a példa szerinti munkafolyamatok időtartama.

3. Pontosság: az emberi erőforrás hibázhat, amely – ezen hibák korrigálásával együtt – jelentős költségnövekedéssel járhat a szervezet számára. Egy felmérés [6] szerint a válaszadók 85%-a számolt be arról, hogy az RPA elérte vagy meghaladta az elvárásait az alkalmazott területen. Természetesen ugyanakkor a pontossági tényezőhöz elengedhetetlenül szükséges az RPA pontos, precíz implementációja a fejlesztők részéről.

A fent részletezetteket meghaladóan még számos további előnnyel járhat az RPA alkalmazása: biztonság növelése, rugalmasság, továbbfejlesztett elemzés, optimalizált erőforráshasználat, automatizált vagy félautomatizált válaszok, meghatározott esemény bekövetkezéséhez kötött triggerek.

2.2. Alkalmazási területek

A különböző folyamatok, eljárások automatizálását a társadalom manapság is alkalmazza és széles körben találkozhatunk automatizált megoldásokkal. Az automatizálás magában

foglalja bármilyen rendszernek a feldolgozási képességét. A különböző, jellemzően összetettebb rendszerek és az emberek integrálása az automatizálás elérése céljából nem triviális feladat, ugyanis legtöbbször az emberi tényezőket igen gyakori előfordulással félreértik, túlzottan leegyszerűsítik vagy figyelmen kívül hagyják a rendszertervezés során [4].

Az RPA néhány alkalmazási területe [5]:

1. Pénzügyi és banki szolgáltatások: számlakezelés, számlakivonat készítés, biztonsági mentések, adatmigráció különböző banki rendszerek között.
2. Gyártás: termékárak összehasonlítása, ERP automatizálás, monitorozás, logisztikai folyamatok automatizálása.
3. Kiskereskedelem: gyártói adatok összegyűjtése a különböző gyártók weblapjairól, elérhető készlet folyamatos frissítése, e-mail értesítések.
4. Egészségügy: biztosítással összefüggő adatok lekérdezése, ellenőrzése, betegnyilvántartás, betegadatok feldolgozása, jelentéskészítés az ellátó személyzetnek.
5. Hírközlés: a versenytársak árazásával kapcsolatos adatok kinyerése, híváslista lekérdezése, ügyfélkapcsolati rendszerek kezelése, számlázás.

2.3. RPA alkalmazása a szoftverfejlesztés munkafázisaiban

A szoftverfolyamat [7] olyan tevékenységek összessége, amelyek egy szoftverrendszer előállításához vezetnek. Tekintettel arra, hogy sokféle szoftverrendszer létezik, nem áll rendelkezésre olyan univerzális szoftverfejlesztési módszer, amely valamennyi szoftver előállítására teljeskörűen alkalmazható. A különböző vállalatoknál alkalmazott folyamat függ a fejlesztendő szoftver típusától, a vásárló vagy használó igényeitől és a szoftver fejlesztésében részt vevők képességeitől is. Bár sokféle különböző szoftverfolyamat létezik, rendszerint mindegyik tartalmazza az alábbi négy alapvető szoftverfejlesztési tevékenységet:

1. Szoftverspecifikáció: meg kell határozni a szoftver funkcionalitását és működésének korlátait.
2. Szoftverfejlesztés: a specifikációnak megfelelő szoftvert kell elkészíteni.
3. Szoftvervalidáció: a szoftvert validálni szükséges. Ennek keretében szükséges megbizonyosodni arról, hogy a lefejlesztett szoftvertermék alkalmas arra, amire tervezték.
4. Szoftver evolúció: a szoftvert a változó vevői igényeknek megfelelően kell fejleszteni.

Ezen munkafázisok önmagukban is összetett tevékenységek, ezért alfeladatokat foglalnak magukban, mint például a követelmények validálása, architektúra tervezése és egységtesztelés [7].

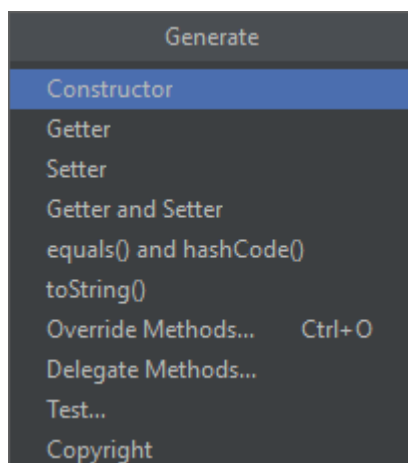
Mivel jelen dolgozat a fenti tevékenységek közül az implementációra kíván összpontosítani, a továbbiakban ezen fázist mutatom be részletesebben.

Talán a legalapvetőbb példa [8] az automatizálásra a különböző keretrendszerek használata. Számos népszerű keretrendszer előre meghatározott többnyire elemi funkciókkal és egységtesztekkel rendelkezik. Például a széles körben használt Maven [9] esetén egyetlen *pom.xml* fájlban megadhatók a projekthez szükséges információk, ideértve a függőségeket és a beépülő modulokat is. Ezeket a keretrendszer letölti és a projektet futtatásra kész állapotra csomagolja, amellyel jelentős időt takarít meg.

A **modell vezérelt fejlesztés** (Model-Driven Development – MDD) [10] a szabványos modellek újra felhasználására épít. A modell vezérelt fejlesztés során a forráskód, a tesztek és a dokumentáció automatikusan generálódnak a különböző modellekből, diagramokból.

Mindezek okán a módszertan alkalmazása növelheti a termelékenységet és csökkentheti a költségeket.

A **kódgenerálást** valószínűleg minden fejlesztő használja. A kódírás közben a figyelmeztetések (warning), hibák (error) valós idejű jelzése mellett a legtöbb integrált fejlesztői környezet (IDE) egyik alapvető funkciója a kódkiegészítés, valamint gyakran alkalmazott programkódrészletek legenerálása.

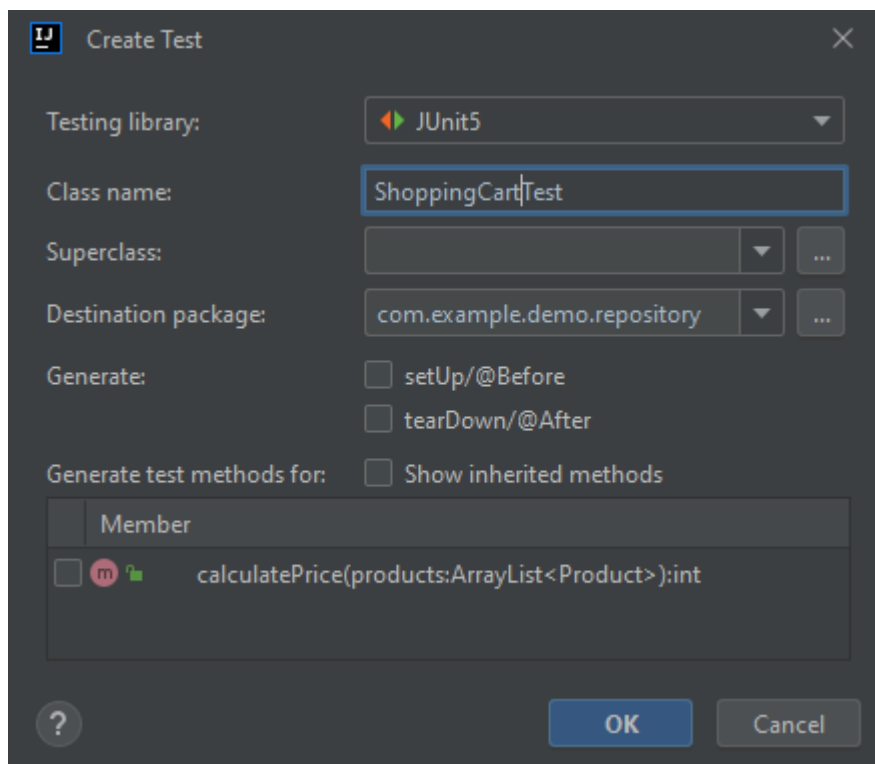


2. ábra. Kódgenerálási lehetőségek (IntelliJ IDEA)

Az 2. ábrán látható képmetszet alapján a *JetBrains IntelliJ IDEA* fejlesztői környezet egy egyszerű objektum orientált JAVA nyelven létrehozott projekthez az alábbi rendszeresen használt megoldásokat támogatja:

- konstruktor legenerálása
- az adattagok értékének beállításához és lekérdezéséhez *getter* és *setter* metódusok (akár külön-külön) generálása
- az adott objektumosztály adatait kiíró gyakran használt *toString()* metódus implementálása
- az őszosztályból vagy az interfészből felüldefiniálni kívánt metódus vázának legenerálása
- egy adott osztályhoz tesztosztály létrehozása.

A tesztosztály generáltatása során (3. ábra) kiválaszthatóak egyebek mellett a tesztelni szándékolt metódusok, valamint a teszteléshez használni kívánt keretrendszer (pl. JUnit5).



3. ábra. Konfigurációs lehetőségek tesztgeneráláshoz (IntelliJ IDEA)

Nyilvánvalóan más fejlesztői környezetek is támogatnak hasonló segítő funkciókat, például az Eclipse-ben is rendelkezésre áll lehetőség a fentiekre.

A szoftverfejlesztés életciklusában a tesztelés is egy olyan folyamat, amelynek során régóta alkalmaznak automatizálást, mivel az automatizált tesztelés jelentősen csökkenti az eljárás időigényét. Az **automatizált tesztelés** [11] alkalmazása esetén nem szükséges minden egyes változtatáskor kézzel bevinni a tesztelni kívánt rendszerbe a szükséges adatokat, mivel a tesztek szükség esetén automatikusan futtathatók. A rendszer gyakorlati alkalmazása és használata alkalmával felmerülő különböző forgatókönyvek szerint lehet vizsgálni az alkalmazás viselkedését. Kiemelten fontos az automatizálás a tesztelésnél, mivel a fejlesztési munkával párhuzamosan számos alkalommal szükséges újra és újra ugyanazokat a funkciókat tesztelni sok esetben akár több platformon egyidejűleg. Számos népszerű tesztelési keretrendszer alkalmas a tesztek eredményéről automatikusan jelentést generálni. [12]

A szoftvertesztelés egyik, talán a leggyakrabban alkalmazott módja az egységtesztelés (Unit testing) [13], amely folyamat során az alkalmazás legkisebb tesztelhető részeit külön-külön és egymástól függetlenül ellenőrzik a megfelelő működés érdekében. Egységtesztelésre

régóta számos keretrendszer áll rendelkezésre a különböző programozási nyelvekhez, talán a legismertebbek például a Java nyelvhez a JUnit, C nyelvhez a CUnit.

Ezekon felül temérdek módszer létezik még a szoftvertesztelésre. Lehetőség van például webböngészőt is automatizálni, amelyre kiváló projekt a Selenium [14]. Ez natív módon használja a böngészőt, ahogyan azt a felhasználó tenné, így hasznos lehetőséget biztosít webalkalmazások és webszolgáltatások tesztelésére.

3. A szoftvertesztelés és a programozói teljesítménymérés mutatói [15]

Egy programozó vagy programozók egy csoportja által elvégzett fejlesztési munkát számos módszer alkalmazásával lehetséges értékelni.

A szoftverfejlesztés és annak oktatása során egyaránt fontos szempont a fejlesztő képessége a magas színvonalú forráskód írására. Elvárhatók a kódolási szabványok betartása, az alpalgoritmusok helyes használata, a kód hatékonysága és redundanciamentessége. A statikus kódelemzés hasznos, mivel a programkódban olyan hibák feltárására alkalmas, melyeket más tesztelési eljárással nehéz megtalálni. Például az alkalmazás lefagyását eredményező null pointer hivatkozás statikus kódelemzés útján kiszűrhető. A statikus kódelemzés történhet kizárólag a forráskód alapján, továbbá a forráskód és a modell együttese alapján is. Előbbi esetében értelemszerűen kizárólag a forráskód kerül felhasználásra, míg utóbbinál a forráskódhoz tartozik egy modell is, amely elő- és utófeltételekkel, továbbá invariánsokkal leírja, hogyan kellene a programnak működnie [16].

A kód minőségének mérésére nincs egyetlen és egységes módszer, azonban létezik néhány kulcsfontosságú alapelv, melyek követése magasabb minőségű programkódot eredményez [17].

3.1. Megbízhatóság

Egy szoftver minősége egyenes arányosságban áll a megbízhatósággal. A termék rendelkezésre állása és a hibák száma befolyásolja legjelentősebben a megbízhatóságot. Ahhoz, hogy minél kevesebb legyen a hiba, a program futása során az előforduló problémák számát szükséges megbecsülni, majd ezen értéket csökkenteni. A problémák száma statikus kódelemzéssel állapítható meg, míg a rendelkezésre állás a meghibásodások közötti átlagidővel (MTBF – Mean Time Between Failures) mérhető [17]. Az MTBF a vizsgálat időtartama és az ezen időintervallum alatti meghibásodások számának hányadosaként számítható több modulból álló, komplex szoftverek tesztelése során [18]. Értelemszerűen minél nagyobb az MTBF értéke, annál megbízhatóbb és magasabb a rendelkezésre állása az adott szoftverterméknek.

3.2. Karbantarthatóság

A karbantarthatóság az a mérőszám, amely megmutatja, hogy milyen hatékonyan (egyszerűen és gyorsan) lehet egy szoftverterméket módosítani. A módosítások magukban foglalják a szoftver javítását, fejlesztését, a környezeti változásokhoz, követelményekhez és funkcionális előírásokhoz történő adaptálását. Karbantartásnak minősül továbbá a speciális támogató személyzet, az üzleti vagy üzemeltetési személyzet, valamint a végfelhasználók által végrehajtott változtatások, emellett ide tartozik még a frissítések, javítások telepítése is [19]. A fenntarthatóságot befolyásolja a szoftver mérete, a struktúrája, a kódbázis komplexitása és a rendszer következetessége is. A fenntartható forráskód biztosítása számos további tényezőn alapul, például a tesztelhetőségen, olvashatóságon és érthetőségen [17].

A forráskód bonyolultságát többféle módszerrel is lehet mérni. Ezek közül a Halstead-féle komplexitási metrikát mutatom be, mely a forráskód szövegelemzésén és információelméleti megfontolásokon alapul.

A Halstead-metrika négy alapmennyiségből indul ki:

- n_1 : különböző operátorok és írásjelek száma,
- n_2 : különböző operandusok száma,
- N_1 : az operátorok és írásjelek összesített száma,
- N_2 : az operandusok összesített előfordulási száma.

Ezek segítségével lehet kiszámítani az alábbi, komplexitást jellemző mérőszámokat [20, 21]:

1. A program szókinccse: a programban található különböző azonosítók számát jelenti, mely az alábbi képlettel kapható meg: $n = n_1 + n_2$.
2. A program hossza: a programban található azonosítók összes előfordulásainak együttes számát jelenti. Képlete: $N = N_1 + N_2$.
3. A program volumene: a szókinccs információtartalma, vagyis a program (mint szöveg) kódolásához szükséges bitek számát jelenti. Képlete: $V = N * n$.
4. A program nehézségi szintje: a program megírásához vagy megértéséhez kapcsolódó mutató. Képlete: $D = \frac{n_1 * N_2}{n_2}$.
5. A program implementálásához szükséges erőfeszítés mértéke arányos a program volumenével és nehézségi szintjével: $E = V * D$.

6. A program implementálásához szükséges idő: tapasztalati kísérletek alapján Halstead megállapította, hogy ha az erőfeszítést elosztjuk 18-cal, akkor megkapjuk a program implementálásához szükséges időt másodpercben: $T = \frac{E}{18}$.
7. Lehetséges hibák száma: $B = \frac{E^{2/3}}{3000}$.

3.3. Tesztelhetőség

A tesztelhetőség mérhető azzal, hogy hány tesztet szükséges egy program teljeskörű teszteléséhez. Ezt a mennyiséget a szoftver összetettsége és mérete határozza meg. A szükséges tesztesetek számának megállapításához használható a ciklomatikus komplexitás, amely a forráskód alapján kiszámítja a lineárisan független útvonalak számát.

A ciklomatikus komplexitás fogalmát Thomas J. McCabe vezette be 1976-os publikációjában [22], melyben bemutatja, hogy a gráfelmélet hogyan alkalmazható programozási szempontból. A ciklomatikus komplexitás a lehetséges kimeneteket köti össze a megfelelő bemenetekkel. Értéke a

$$CYC = e - n + 2 * p$$

formulával kapható meg, ahol

- e a gráf éleinek száma (vezérlés átadása),
- n a gráf csúcsainak száma (szekvenciális utasítások csoportja),
- p az összefüggő komponensek száma (kilépési pontok száma).

3.4. Hordozhatóság

A hordozhatóság azt deklarálja, hogy egy szoftver mennyire platformfüggetlen, azaz hogyan támogatja a különböző környezetekben való futtatást ugyanaz a kód és mekkora mennyiségű energiabefektetéssel alakíthatók át a nem platformfüggetlen egységek szükség esetén úgy, hogy más környezetben is működőképes legyenek. Akkor hordozható egy része a szoftvernek, ha az új platformra való adaptálás alacsonyabb költségigényű, mint az adott egység újrfejlesztése [23]. Környezet vagy platform alatt értendő egyebek mellett az operációs rendszer, a processzor és más fizikai környezet [24]. A hordozhatóságot a hordozási és újjáépítési költségek függvényével lehet értelmezni, amelyet egy adott célkörnyezethez viszonyítunk [23].

3.5. Újrafelhasználhatóság

Az újrafelhasználhatóság a szoftverek azon jellemzője, hogy a meglévő szoftver, komponens vagy programkód milyen mértékben használható fel újra más alkalmazások előállításánál [24]. Ezzel jelentős idő – és ebből következően költség – takarítható meg a fejlesztés során, mivel a már meglévő, ismételten hasznosítható komponenseket nem szükséges újból kifejleszteni.

Az újrafelhasználhatóságra építkeznek szoftverfejlesztési módszertanok is, például a komponens alapú fejlesztés paradigma szerint megkíséreljük újrafelhasználható komponensekből felépíteni az elkészítendő szoftverterméket. Az újrafelhasználás-orientált modell a meglévő és elérhető újrahasznosítható komponensekre, illetve azok egységes szerkezetbe történő integrációjára támaszkodik [16].

4. A GitHub alkalmazása RPA eszközként

A továbbiakban bemutatom, hogy az automatizált programkód kiértékelését a Miskolci Egyetem Általános Informatikai Intézeti Tanszéke által a mintatanterv szerinti első évfolyamon oktatott Programozás alapjai című tárgy gyakorlati feladataihoz hogyan valósítottam meg.

4.1. Hallgatói csoport és a kurzus jellemzése

A Programozás alapjai tantárgy teljesítése a mérnökinformatikus, gazdaságinformatikus, programtervező informatikus és üzemmérnök informatikus alapképzési szakok képzési programjában az első tanulmányi félévre került előírásra. A tárgyra jelentkező hallgatók jelentős része tehát az egyetemi tanulmányai kezdetén áll, míg kis részük ismételten veszi fel a tárgyat, mivel az előző évben nem sikerült teljesíteniük a követelményeket.

A programozás oktatása nem kötelező a középiskolákban, ezért a hallgatók jelentős része ezen kurzus alkalmával találkozik először programozási feladatokkal, tehát fontos szempont, hogy számukra is megfelelő lehetőség álljon rendelkezésre a programozással történő ismerkedésre és a gyakorlásra. A programozás ugyanis csak gyakorlás útján tanulható meg, de önálló órarendi keretek között erre nincs elég idő. Az otthoni gyakorlás gátja pedig gyakran a segítség hiánya.

4.2. Előkészületek

Az elkészítendő munka egy olyan megoldás kialakítása, amely automatikusan kiértékeli a hallgatók által megoldott gyakorlati programozási feladatokat. Célunk egy, az otthoni gyakorlást támogató módszer kidolgozása, amely biztosítja a tanulók számára a folyamatos visszajelzést.

Mivel egy programozási feladat tekintetében a szintaktikai ellenőrzést megvalósítja a fordítóprogram, főként a szemantikai vizsgálatra kellett a hangsúlyt fektetni. Az automatizálendő feladat jelentős részét ebből következően az tette ki, hogy automatikusan ellenőrzésre kerüljenek a hallgatók által feltöltött megoldások abból a szempontból, hogy megfelelnek-e a feladatkiírásnak.

Mindenekelőtt nélkülözhetetlen volt gyűjteni a tárgyhoz tartozó gyakorlati feladatokat, illetve több esetben is előfordult, hogy azokon a tesztelhetőség biztosítása céljából minimális

átalakításokat kellett végezni, valamint néhány új feladvány kitalálása is szükségszerűvé vált. A feladatkiírások összegyűjtését és szükség szerinti átszövegezését követően indokolt volt a feladatokat megoldani, valamint az értékelésükről gondoskodni tesztesetek segítségével. Megoldást kellett találni arra is, hogy a hallgatói eredmények egységes szerkezetben, könnyen átlátható és értelmezhető formában legyenek kinyerhetők a kurzus oktatói számára.

4.3. Alkalmazott technológiák és szolgáltatások

Olyan megoldást volt szükségszerű találnom, ahol a feladatok kiadása a hallgatók részére egységesen és hatékonyan megoldható és a felületen a hallgatók az oktató által megállapított határidőre be tudják küldeni és a beküldéssel egyidejűleg a kiértékelés is megvalósul.

A feladatokat a GitHub [25] verziókövető rendszerben gyűjtöttem össze, míg az értékelés szabályrendszerét a GitHub Classroom-ban [26] adtam meg az egyes feladatokhoz.

A GitHub [25] egy számos funkcióval rendelkező, alapvetően forráskódok tárolására, verziókövetésére és kollaborációra alkalmazható platform. Segítségével a fejlesztők bárholnan együttműködhetnek különböző projekteken. Támogatja szervezetek (organization) létrehozását is, ahol együttműködést biztosít egyszerre több projekten is kifinomult biztonsági és adminisztrációs funkciókkal. A GitHub számos elérhető funkciója mellett lehetőséget kínál programozás jellegű tárgyak, tanfolyamok oktatására is. A GitHub Classroom nagymértékű szabadságot ad az oktatók kezébe: hallgatókat rendelhetnek hozzá, tetszőleges feladatokat hozhatnak létre és adhatnak ki a tanulók számára. A feladatok automatikus kiértékelése is beállítható, amellyel a feladvány hallgató által történő feltöltésekor automatikusan megtörténik a kiértékelés és visszajelzést kap a tanuló az eredményről. Természetesen sikertelen vagy nem maradéktalanul megfelelő eredmény esetén megkísérelhető a feladvány ismételt beadása.

Az összetettebb, függvények megírását is megkövetelő feladatok teszteléséhez felhasználtam a CUnit [27] egységtesztelési keretrendszert is, amelyben azt vizsgálom, hogy egy adott feltétel teljesül-e a program egy adott pontján. Ehhez az egyes függvények visszatérési értékeire vonatkozóan állításokat (assertion) fogalmazok meg, amelyek automatikusan kiértékelésre kerülnek.

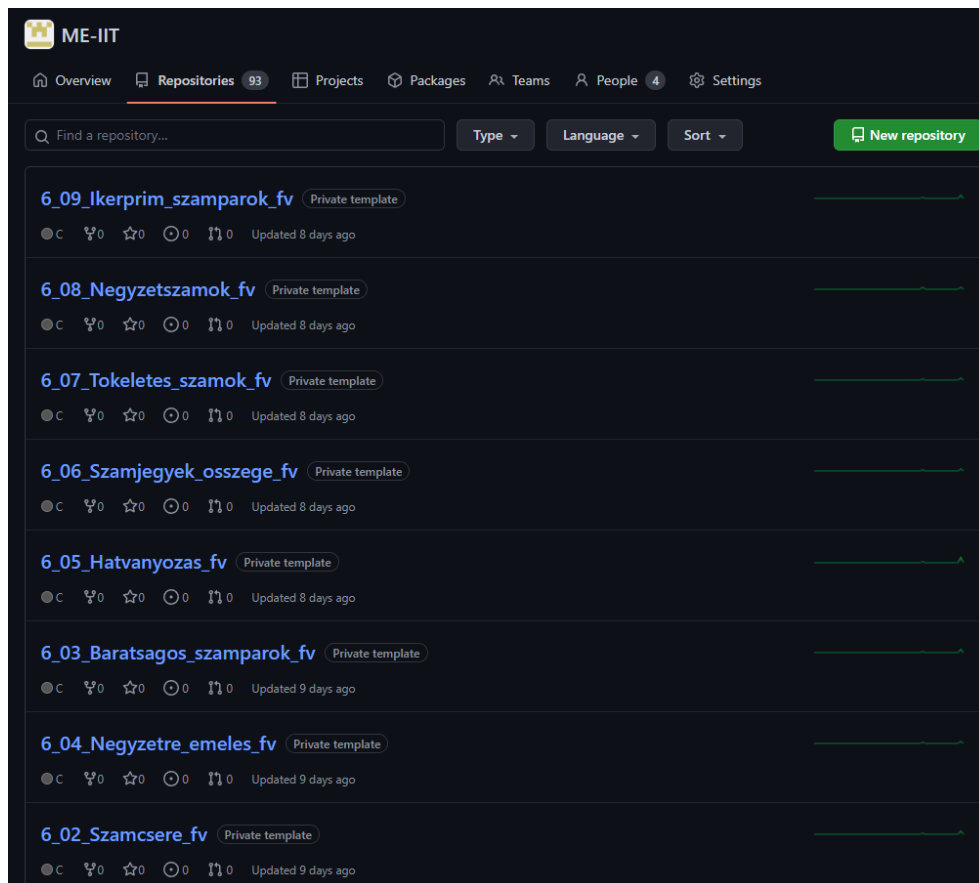
A fentiek alkalmazása azért is előnyös, mert igény esetén maximális rugalmasságot biztosítanak a feladatok, a kiértékelés módosításával, törlésével és hozzáadásával összefüggésben is.

4.4. Megvalósítás

A GitHub-ra már a munka megkezdése előtt regisztráltam, így arra nem volt már szükség. Ezért a C programozási feladatkiírások gyűjtésével kezdtem meg az érdemi munkát, amelynek legfőbb forrásai a tárgy előadója által rendelkezésemre bocsátott gyakorlati példák voltak. Ezek között kezdetben olyan rendkívül triviális feladványok találhatók meg, mint például egy kör területének és területének kiszámítása felhasználó által megadott sugárérték alapján, majd a különböző alapalgoritmusok megértését elősegítő feladatok következnek. Rendelkezésre állnak a tömb-, sztring- és struktúrakezelés elsajátítására és gyakorlására egyaránt feladatok. Léteznek olyan példák is, amelyek megoldásához egy kezdő számára előfordulhat, hogy némiképp intenzívebb gondolkodás lehet szükséges.

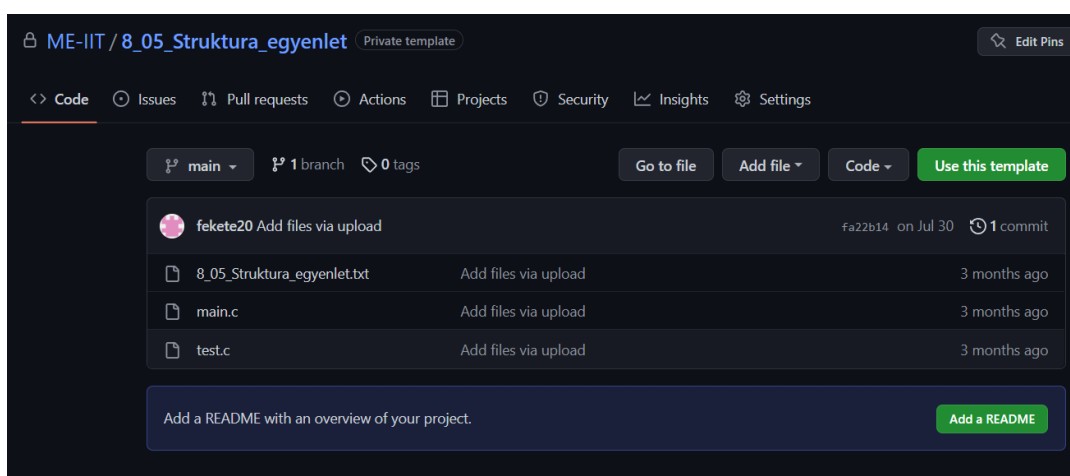
A feladatkiírások véglegesítését követően szükséges volt megoldanom azokat annak érdekében, hogy a rendszer működését megfelelően legyen lehetőségem ellenőrizni és a felmerülő problémákat az éles használatot megelőzően korrigálni. Szükséges volt továbbá megtervezni, hogy milyen teszteket kell az egyes feladatok automatizált kiértékeléséhez alkalmaznom.

A feladványok áttekinthető nyilvántartása céljából a GitHub felületén egy szervezetet (organization) hoztam létre, amelyben külön adattárakba (repository) szerveztem az egyes feladatokat.



4. ábra. ME-IIT organization (GitHub)

A repository-k szerkezete kvázi egységes valamennyi példa esetén, ugyanakkor az egyes fájlok tartalma eltérően igazodik a feladványhoz. Ezeket alkalmazom majd a hallgatók részére kiadandó feladatoknál sablonként. A lentebb megtalálható 5. ábrán egy másodfokú egyismeretlenes egyenlet komplex számok halmazán történő megoldását implementáló program megírását megkövetelő feladathoz tartozó jegyzék látható.



5. ábra. Konkrét feladathoz tartozó sablonjegyzék

Az 5. ábrán látható jegyzékben lévő fájlok az alábbiakat tartalmazzák:

1. A feladatkírás szövege egyszerű szövegfájlban, *txt* formátumban.
2. A feladathoz tartozó kezdeti kód a *main.c* állományban, amely minden esetben tartalmazza a *main* függvény vázát. Amennyiben a feladat megoldásához függvényírás szükséges, úgy ezen fájlban a különböző függvények visszatérési típusát, nevét (azonosítóját) és paraméterlistáját tartalmazó deklarációját is megadtam.
3. Függvényírást kérő feladatok esetén a függvényekre CUnit-ban megírt egységtesztek a *test.c* állományban. Amennyiben az adott feladatot kizárólag a *main* függvényben kell megoldani, úgy ilyen fájl nem képezi részét a feladványt tartalmazó jegyzéknek tekintettel arra, hogy ilyen esetben a tesztelés másképpen történik meg.

A feladatok organization-be való felvitele közben párhuzamosan végeztem el azoknak a classroom-ban történő létrehozását és konfigurációját.

A létrehozás során valamennyi feladvány tekintetében megadtam a konkrét feladathoz tartozó, organization-ben lévő sablont (template repository). Emellett opcionális lehetőségként a classroom lehetőséget szolgáltattam arra is, hogy bár meglehetősen szűk választékkal fejlesztői környezetet jelöljek ki a kiírt feladathoz, amellyel a hallgatónak egy egérgattintással megnyílik és betöltődik a feladat és megkezdhető a megoldása.

Ezt követően van lehetőség az automatizált kiértékeléshez a teszteket megadni. Ennek keretében az alábbi lehetőségeket kínálja a GitHub:

- Input/Output teszt
- Megadott parancs lefuttatása parancssorban (Run command)
- Java, NodeJS, Python, C vagy C++ tesztesetek futtatása.

Megjegyzendő, hogy a Java, NodeJS, Python, C és C++ tesztesetek futtatása gyakorlatilag a parancsfuttatási lehetőséggel (Run command) egyeznek meg azzal az eltéréssel, hogy a GitHub Classroom a felsorolt programnyelvek esetében használatos alapvető parancsok futtatását (például NodeJS esetén *npm test*) írja elő. A rendszerbe feltöltött feladatok kiértékelésének automatizálásához elsősorban az Input/Output tesztelést, valamint az összetettebb, függvényírást megkövetelő feladatok vonatkozásában a CUnit egységteszt keretrendszer telepítéséhez és futtatásához a parancsfuttatási lehetőséget használtam. Természetesen egy feladathoz több tesztet is felvihető, a tesztek típusai vegyíthetők, így a

szemantikai kiértékelés során valamennyi feladatkiírásban szereplő követelmény vizsgálata automatizálható.

Input/Output tesztelés felvitele során az alábbi 6. ábrán látható párbeszédablakot szükséges kitölteni.

The screenshot shows a web-based form titled "Input/Output test case". It has several input fields and a dropdown menu. The fields are: "Test name" with the value "Not0_test", "Setup command (optional)" with the value "gcc main.c -o main -lm", "Run command" with the value "./main", "Inputs" with the value "1802, 1850", "Expected Output" with the value "12", "Comparison" with a dropdown menu showing "included", "Timeout (minutes)" with the value "10", and "Points (optional)" with the value "1". At the bottom of the form is a green button labeled "Save test case".

6. ábra. *Input/Output autograding teszt megadása (GitHub Classroom)*

A fenti 6. ábrán egy olyan, a gyűjteményből kiragadott feladathoz tartozó tesztelés látható, amely két évszám bekérése után megszámolja az intervallumba eső szökőévek számát és kiírja a szabványos kimenetre. A feladvány trivialitása okán a megoldás ellenőrzésének automatizálásához mindössze 2 esetet volt szükséges vizsgálni:

1. Az eredmény nulla, vagyis a megadott intervallumon belül nem volt szökőév.
2. Az eredmény nem nulla, ekkor ki kell írni az intervallumba eső szökőévek számát.

A 6. ábrán az utóbbi eshetőség vizsgálatához szükséges tesztelés látható. Ennek létrehozásakor az alábbiakat kell megadni:

- A teszt nevét, amellyel később visszakereshető és igény esetén módosítható.
- Opcionálisan egy beállítási parancsot (setup command). Ez egyszerű Input/Output tesztelés esetén a hallgató által beküldött forráskód fordítását jelenti parancssorban.

- Futtatási parancsot, amellyel a gépi kódra lefordított forráskód futtatható.
- A feladat jellegéhez igazodóan ugyancsak opcionálisan a program bemenetét. Ezt pontosan olyan formátumban kell rögzíteni, amilyenben az alkalmazás futtatásakor a parancssorban is meg kell adni az inputokat. Az automatizált kiértékelés során a GitHub az itt definiált bemeneti értékeket fogja megadni a programnak. Lehetőség van sortöréssel, több sorban is bevinni a mező tartalmát. Ilyenkor az értékelés folyamán automatikusan ennek megfelelően adja meg a programnak a bemeneti értékeket, vagyis a több soros bemenetet kérő feladványok értékelése is megfelelően automatizálható.
- Az elvárt kimenetet, amely megadható reguláris kifejezésekkel, továbbá kiválasztható, hogy pontosan a mezőben megadott értékeket (exact) kell tartalmaznia a program kimenetének vagy elegendő szerepelnie abban (included).
- A teszt futtatásának maximális idejét percben mérve. Ennek hibásan megírt program vagy nem megfelelően elkészített teszteset esetén van érdemi jelentősége a rendszer terhelésének csökkentésére, mivel az itt rögzített idő eredménytelen leteltét követően időtúllépésre (timeout) hivatkozva megáll az értékelés.
- Végül az oktató által tetszőlegesen megadható, hogy hány pontot ér a feladatmegoldás tekintetében, ha az adott teszteset eredménye sikeres.

Azon feladatok esetén, amelyeknél függvényírás történik, a CUnit-ban megírt tesztek automatikus végrehajtásához a *Run command* lehetőséget is használtam.

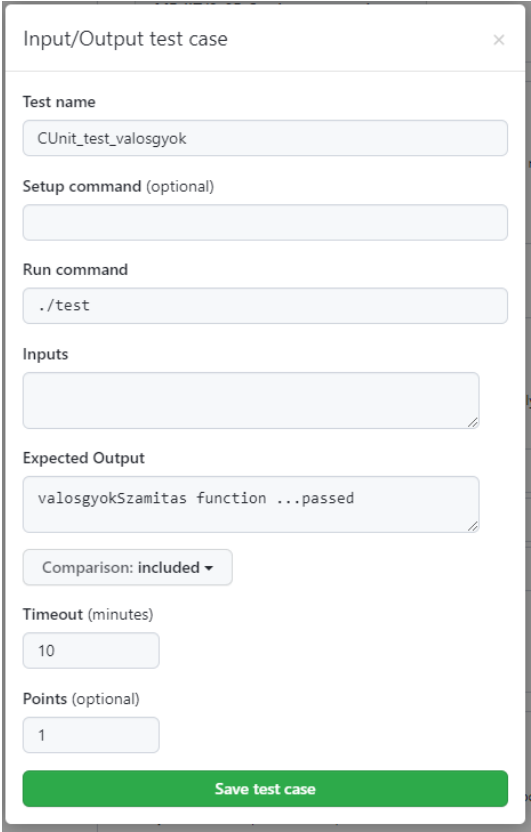
The image shows a 'Run command' dialog box with the following fields and values:

- Test name:** CUnit_install_compile
- Setup command (optional):** sudo apt-get install libcunit1 libcunit1-doc libcunit
- Run command:** gcc test.c -o test -lm -lcunit
- Timeout (minutes):** 10
- Points (optional):** 0
- Button:** Save test case

7. ábra. CUnit beállítása parancsfuttatás automatizálásával

A 7. ábrán előbb a keretrendszer telepítését állítottam be a `sudo apt-get install libcunit1 libcunit1-doc libcunit1-dev` paranccsal, majd az adott feladványhoz tartozó jegyzékben megtalálható, a dolgozatom 5. ábráján is látható `test.c` állomány fordítását írtam elő a `gcc test.c -o test -lm -lcunit` paranccsal. Tekintve, hogy itt a feladat érdemi értékelése előtt kizárólag a CUnit-ban megírt tesztesetek előkészítése valósul meg automatikusan, nem került beállításra értelmezhető pontszám.

Ezután a különböző függvényeket egyenként Input/Output teszteset beállításával egyszerűen lehet vizsgálni. Amennyiben egy függvényhez tartozó valamennyi kritérium (assertion) teljesül, a CUnit kimenete a vizsgált függvény nevéhez igazodóan „*valosgyokSzamitas function ...passed*” formátumban jelenik meg, így a lentebb megtekinthető 8. ábrán láthatóan mindössze a *Run command* lehetőséggel már lefordított teszteset futtatását, valamint az elvárt kimenetet szükséges megadni.



The screenshot shows a window titled "Input/Output test case" with a close button (X) in the top right corner. The window contains several input fields and a dropdown menu. The "Test name" field contains "CUnit_test_valosgyok". The "Setup command (optional)" field is empty. The "Run command" field contains "./test". The "Inputs" field is empty. The "Expected Output" field contains "valosgyokSzamitas function ...passed". Below this is a dropdown menu labeled "Comparison: included". The "Timeout (minutes)" field contains "10". The "Points (optional)" field contains "1". At the bottom of the window is a green button labeled "Save test case".

8. ábra. CUnit tesztelés sikerességének vizsgálata

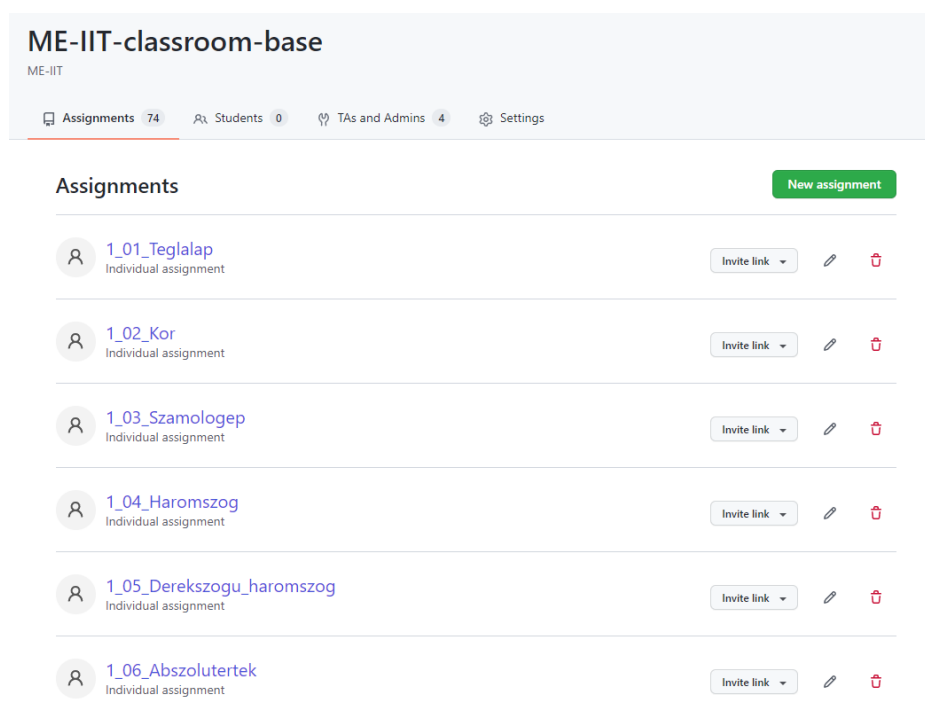
A jelen dolgozat 3. fejezetében kifejtett, a programkód minőségének automatizált vizsgálatára alkalmas statikus kódelemzési lehetőségek használatára és a minőségi mutatók kiszámítására is használhatóak a fenti funkciók, mivel számos parancssorban futtatható, és ezáltal a GitHub eszközeivel automatizálható program áll (pl. *cppcheck*, *pmccabe* stb.)

rendelkezésre ezekhez. Ugyanakkor figyelemmel arra, hogy mindezek eredményei nagymértékben változatosak, nem beszédesek és az oktatók által jelenleg nehezebben, egyesével nyerhetőek ki, ezek a jelen kutatásban nem kerültek megvalósításra.

4.5. Működés a gyakorlatban

4.5.1. Bemutató oktatói nézőpontból

A GitHub Classroom felületét oktatói szerepkörben megnyitva a 9. ábrán látható felület tárul elénk.



9. ábra. *GitHub Classroom indítóképernyője oktatói szerepkörben*

Itt lehetősége van az oktatóknak megtekinteni, módosítani, törölni a rendszerben lévő feladatokat, de akár újat is hozhatnak létre vagy további oktatókat, adminisztrátorokat hívhatnak meg. Minden feladványhoz tartozik egy link, amelyet az oktató megküldhet a hallgatóság részére, így válik lehetségessé a hallgatók adott feladathoz való hozzárendelése. A link segítségével történő meghívás lehetősége feladványonként tetszőlegesen aktiválható vagy deaktiválható.

Az egyes feladatokhoz tartozó eredményeket a feladvány nevére történő egérgépettel van lehetőség megtekinteni (10. ábra), amelyek CSV fájlban is exportálhatók. A hallgatók által

beküldött megoldások is egyetlen kattintással letölthetők, illetve van lehetőség visszacsatolás küldésére is.

9_02_Csokolade

The screenshot shows the GitHub Classroom interface for an assignment titled '9_02_Csokolade'. At the top, it indicates 'Individual assignment', 'Active' status, and 'VS Code' as the IDE. A URL bar shows 'https://classroom.github' with 'Edit' and 'Download' buttons. Below this, a summary box on the left shows 'Accepted students: 1', 'Assignment submissions: 0', and 'Passing students: 1/1' with a green progress bar. To the right is a search bar 'Search by GitHub username or student identifier'. Below the search bar, a table titled 'Total students' lists student 'fekete20' with a green checkmark, 'Latest commit passed', '3/3' progress, and '1 commit'. It also shows 'Autograding' and 'Repository' icons.

10. ábra. *GitHub Classroomban lévő feladathoz tartozó eredményeket összesítő képernyő*

Amennyiben az oktató meg kívánja tekinteni egy adott hallgató megoldását és próbálkozási kísérleteit, úgy erre is közvetlenül van lehetősége a fenti nézetből. Ezt meghaladóan az automatizált értékelés részlépései és részeredményei is hasonlóan előhívhatók, mérőszámok és statikus kódelemzés esetén ezek eredményei is innen lennének kiolvashatóak. A hallgatói programkód közvetlen integrált fejlesztői környezetben való megnyitása is egyetlen mozdulattal megvalósítható, azonban a rendszer ezen célra egyelőre csak a Visual Studio Code IDE-t támogatja, amelyhez külön kiegészítő telepítése és konfigurációja szükséges a C programozási nyelvű feladatok kezeléséhez.

A GitHub Classroom-ba felvitt feladatokat az azokhoz tartozó, automatizált tesztek is magában foglaló valamennyi beállítással együtt lehetséges felhasználni bármilyen másik Classroom-ban is. Ez abban az esetben lehet hasznos, ha a nagylétszámú hallgatói csoportot fel kell osztani több kisebb csoportra, amely csoportok gyakorlatvezetői eltérő személyek. Így tehát minden gyakorlatvezetőnek csak a saját csoportjának feladataival, eredményeivel szükséges törődnie anélkül, hogy a feladatok újrarögzítése lenne szükséges a rendszerben.

4.5.2. Bemutató hallgatói szemszögből

Feltételezve, hogy a hallgató előzőleg már regisztrált saját magának egy accountot a GitHub-hoz, első lépésként egy linket kap a feladathoz az oktatótól, amelynek segítségével tudja azt elfogadni. Az elfogadást követően rövid várakozással a 11. ábrán látható felület kerül megjelenítésre.



You're ready to go!

You accepted the assignment, 9_02_Csokolade.

Your assignment repository has been created:

https://github.com/ME-IIT/9_02_csokolade-fekete20

We've configured the repository associated with this assignment ([update](#)).



Note: You may receive an email invitation to join [ME-IIT](#) on your behalf. No further action is necessary.

11. ábra. A feladat elfogadását követően megjelenő hallgatói felület

Látható, hogy a feladatokat tartalmazó ME-IIT organization-ben létrejött egy egyedi repository, amelyben a hallgató dolgozhat a feladaton. Itt lenne lehetőség a *Visual Studio Code*-ban egy kattintással megnyitni a feladatot, ugyanakkor a fejlesztői környezet sajátosságai okán és nem utolsósorban figyelemmel a tárgyat hallgató jellemzően elsőéves hallgatók számára a konfigurációval járó vélt vagy valós nehézségek elkerülése miatt a jegyzék letöltése, majd a szövegfájlban lévő feladatkiírás és a *main.c* állomány tetszőleges alkalmazással való megnyitását javasoltuk. A visszajelzések alapján a hallgatók többnyire a programozáshoz *CodeBlocks* IDE-t választották. A feladatkiírás megtekintésére bármely egyszerű, akár az operációs rendszerbe integrált szövegszerkesztő is használható.

9_02_Csokolade.txt - Notepad

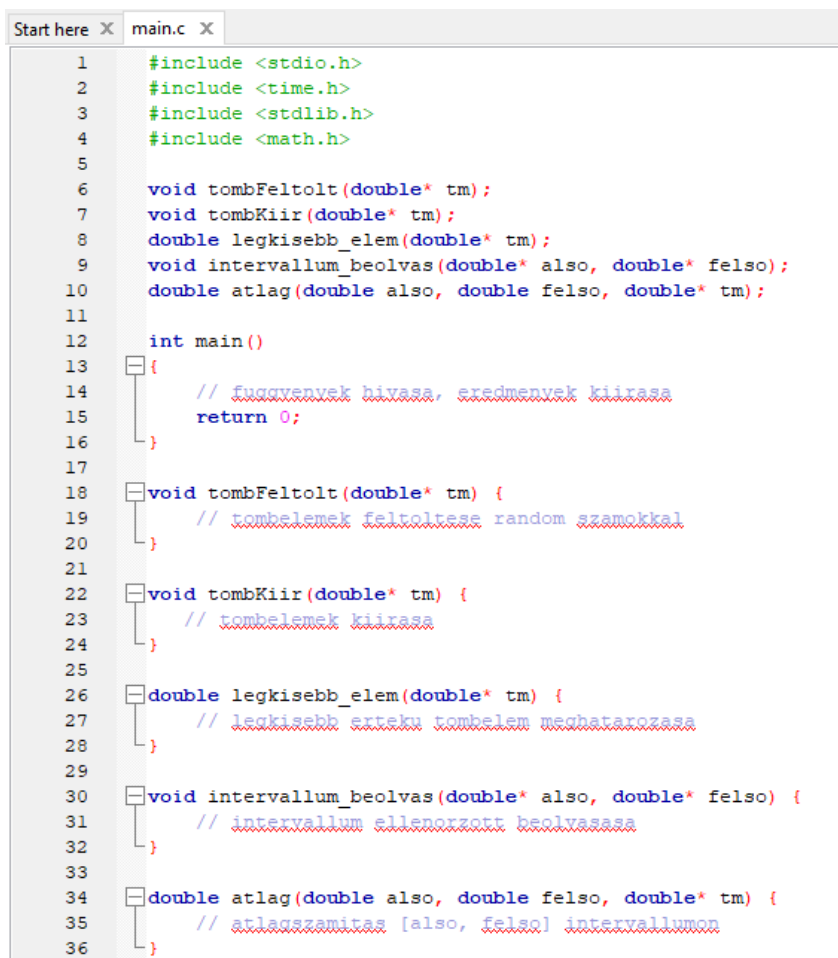
File Edit Format View Help

Írjon C programot, amely 10 adat tárolására alkalmas egydimenziós tömbben nyilvántartja egy csokigyárban az elkészült 100 grammos csokoládé táblák valós tömegét.
Az alábbi részfeladatokat külön függvényben valósítsa meg:
Töltse fel a tömböt 95 és 105 közé eső véletlen valós értékekkel 2 tizedesjegy pontossággal.
Írja ki a tömb elemeit.
Válassza ki a 100 grammot meghaladó értékek közül a legkisebbet.
Olvasson be ellenőrzött módon egy intervallumot a 95 és 105 közötti tartományban. (Pl: 98-100). Az ellenőrzött beolvasás keretében ellenőrizze, hogy a megadott intervallum alsóhatára kisebb legyen a felsőhatárnál és azt is, hogy valóban numerikus értéket adott-e meg a felhasználó. Hiba esetén írassa ki, hogy "Hibas input".
Számítsa ki a beolvasott intervallumba eső tömbelemek átlagát.

A megoldáshoz használja fel a kódvázat, ügyeljen arra, hogy a függvények visszatérési típusa, neve (azonosítója) és paraméterlistája nem térhet el az előírtaktól.

12. ábra. Feladatkiírás megjelenítése szövegszerkesztőben

A fenti 12. ábrán olvasható feladatkiíráshoz tartozó kezdeti kódot tartalmazó *main.c* állomány tartalma az alábbi 13. ábrán tekinthető meg. Ezt szükséges megfelelően kiegészítenie a hallgatónak a feladat teljesítéséhez.

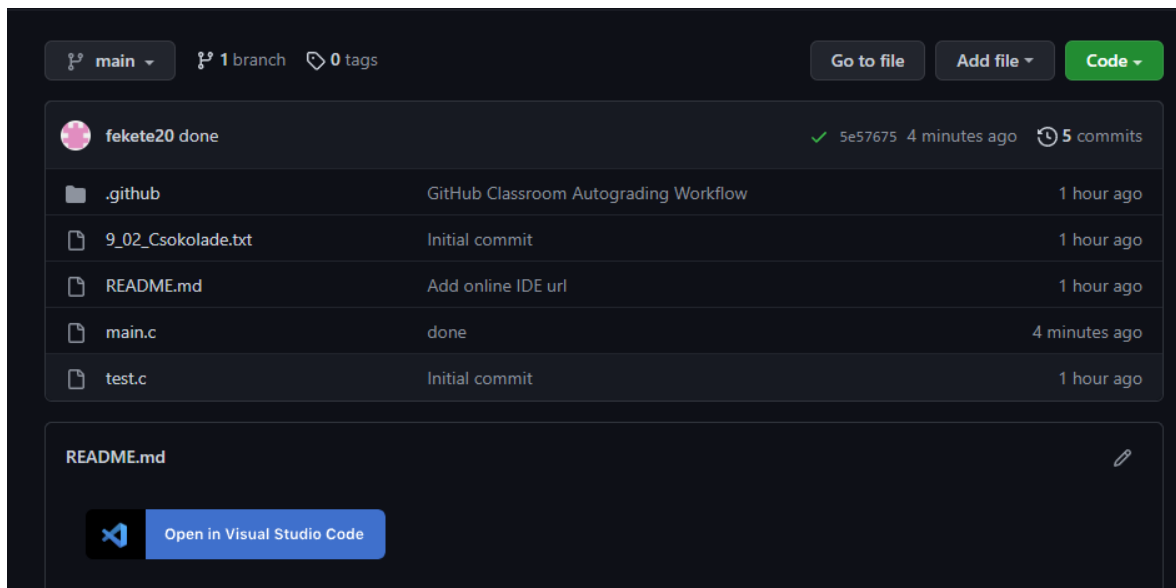


```
1  #include <stdio.h>
2  #include <time.h>
3  #include <stdlib.h>
4  #include <math.h>
5
6  void tombFeltolt(double* tm);
7  void tombKiir(double* tm);
8  double legkisebb_elem(double* tm);
9  void intervallum_beolvas(double* also, double* felso);
10 double atlag(double also, double felso, double* tm);
11
12 int main()
13 {
14     // függvények hívása, eredmények kiírása
15     return 0;
16 }
17
18 void tombFeltolt(double* tm) {
19     // tombelemek feltöltése random számokkal
20 }
21
22 void tombKiir(double* tm) {
23     // tombelemek kiírása
24 }
25
26 double legkisebb_elem(double* tm) {
27     // legkisebb értéku tombelem meghatározása
28 }
29
30 void intervallum_beolvas(double* also, double* felso) {
31     // intervallum ellenőrzött beolvasása
32 }
33
34 double atlag(double also, double felso, double* tm) {
35     // atlagsszámítás [also, felso] intervallumon
36 }
```

13. ábra. Kezdeti forráskód (CodeBlocks IDE)

Miután a hallgató megoldotta a kiadott feladatot, szükséges feltöltenie az azonosítójához generált, 5. ábrán bemutatott repository-ba, amelyre a modernebb fejlesztői környezetekből közvetlenül is nyílik lehetőség.

A feltöltést követően automatikusan lefutnak az adott feladványhoz beállított autograding tesztek, és a sikeres teljesítés esetén egy zöld pipa látható a felületen (14. ábra). Erre kattintva a 15. ábrán prezentált formában részletesen megvizsgálhatók a lefutott tesztek eredményei a konkrét értékekkel is, amelyek különösen hibás megoldás esetére jelentenek segítséget a hallgatóknak.



14. ábra. Sikeresen teljesített feladathoz tartozó repository

A lenti 15. ábrán elérhető autograding workflow logban lennének láthatók a kódminőség jellemzésére használható mutatók értékei is.

```

77
78 Run Summary:  Type  Total   Ran Passed Failed Inactive
79               suites    1     1   n/a     0       0
80               tests     1     1     1     0       0
81               asserts    2     2     2     0     n/a
82
83 Elapsed time =  0.000 seconds
84
85 ✓ CUnit_test
86
87 📁 Input_test_char
88
89
90
91 103.510000
92 95.040000
93 97.940000
94 95.500000
95 99.590000
96 98.920000
97 99.200000
98 103.690000
99 103.000000
100 97.520000
101 Legkisebb elem: 95.040000
102 Adja meg az intervallumot vesszovel elvalasztva:
103 Hibas input
104 Adja meg az intervallumot vesszovel elvalasztva:
105 Atlag: 97.672857
106
107 ✓ Input_test_char
108
109 📁 Input_test_interval
110

```

15. ábra. Autograding workflow log

5. Konklúzió

Jelen tudományos diákköri dolgozatomban a szakirodalom áttekintése segítségével bemutattam a különböző munkafolyamatok automatizálására alkalmazott RPA fogalmát, jellemzőit és egy példát az üzleti életben történő használatára. A dokumentum további részében azt ismertettem, hogy a GitHub Classroom által biztosított szolgáltatásokat hogyan állítottam az RPA szolgálatába, megkönnyítve ezzel egy nagy létszámú programozást oktató egyetemi kurzussal összefüggő, az oktatókat terhelő értékelési munkát.

Az általam alkalmazott konfiguráció automatizáltan ellenőrzi és értékeli a hallgatók részéről beküldött megoldásokat. A rendszerben jelenleg 74 feladat áll rendelkezésre kiadható állapotban, ezek mindegyikéhez megvalósítottam az értékelés automatizálását összesen 252 db tesztesettel. Átlagosan egy feladat kiértékelése a feltöltéstől számított 15 másodpercen belül automatikusan elkészül. A fejlesztésem eredményét a jelenleg folyamatban lévő tanulmányi félévben 115 hallgató és 4 oktató használja. Feladatonként átlagosan 4 beküldés (feltöltés) történik a hallgatóság részéről, ezek közel 90%-a munkaidőn kívül (jellemzően hétköznapi este és hétvégén) érkezik be.

Az oktatók és a tanulók számára egyaránt többféle előnnyel jár a módszer alkalmazása: egyrészt rugalmasságot ad a hallgatóknak, hiszen bármikor és bármennyi alkalommal beküldhetők a megoldások. Másrészt támogatja a távmunka lehetőségét, hiszen az egyes feladatok kiértékelése automatikus, így nem válik szükségessé az oktatók megkérése egy adott probléma megoldásának elmagyarázásához. Az oktatóknak elsősorban azért kedvezőbb, mert a több mint 100 hallgató számos feladatának javítására, mint rutinszerű munkára kevesebb időt kell fordítani. Az automatikus tesztek a gyakran előforduló hibákat kiszűrik, ebből következően kevesebb egyéni konzultációra van szükség. Amennyiben egy oktató egy feladat kiértékelésével átlagosan 5 percet tölt el, úgy a 115 hallgató 74 feladatának javítása összesen közel 710 munkaórával járna. Ezzel szemben a feladatonként átlagosan 15 másodperc alatt végbemenő automatikus értékelés időigénye nem haladja meg a 36 órát.

6. Conclusion

In this research, I presented the concept and characteristics of RPA used to automate various workflows and an example of its use in business through a literature review. In the rest of the paper, I described how I applied the services provided by GitHub Classroom to the service of RPA, facilitating the evaluation workflow burdening the instructors for a large university course on programming.

The configuration I applied automatically checks and evaluates the solutions submitted by students. The system currently has 74 assignments which can be issued, for each of which I have implemented the automation of the evaluation with a total of 252 test cases. On average, the evaluation of an assignment is automatically completed within 15 seconds of upload. The result of my development is used by 115 students and 4 instructors in the current semester. There is an average of 4 submissions (uploads) per assignment from students, with almost 90% of which are received outside working hours (typically weekday evenings and weekends).

For both teachers and students, there are several advantages to using this method: on the one hand, it gives flexibility for students, as they can submit their solutions at any time and any number of occasions. On the other hand, it supports remote working, since the evaluation of each task is automatic, so there is no need to ask the instructors to explain the solution. For the instructors, it is mainly advantageous because less time is needed to assess more than 100 student's assignments as routine work. Automatic tests filter out common mistakes, consequently fewer individual consultations are needed. If an instructor spends an average of 5 minutes evaluating an assignment, it would take almost 710 man-hours the assessment of 74 assignments for 115 students. In contrast, the time required for automatic assessment, which takes on average 15 seconds per task, does not exceed 36 hours.

7. Irodalomjegyzék

- [1] CHROMJAKOVÁ, Felicitá; BOBÁK, Roman; HRUSECKA, D. Production process stability—core assumption of INDUSTRY 4.0 concept. In: IOP Conference Series: Materials Science and Engineering. IOP Publishing, 2017. p. 012024.
- [2] VAN DER AALST, W. M. P.; BICHLER, M.; HEINZL, A. Robotic process automation. Bus Inf Syst Eng 60 (4): 269–272. 2018.
- [3] Ozge Doguc: Robot Process Automation (RPA) and Its Future, In: Advances in E-Business Research, 2020. DOI:10.4018/978-1-7998-1125-1.ch021, URL: https://www.researchgate.net/profile/Ozge-Doguc-2/publication/338302068_Robot_Process_Automation_RPA_and_Its_Future/links/5f5772f592851c250b9d23ad/Robot-Process-Automation-RPA-and-Its-Future.pdf
- [4] Somayya Madakam, Rajesh M. Holmukhe, Durgesh Kumar Jaiswal: The Future Digital Work Force: Robotic Process Automation (RPA), In: JISTEM J.Inf.Syst. Technol. Manag. 16, 2019. DOI: <https://doi.org/10.4301/S1807-1775201916001> URL: <https://www.scielo.br/j/jistem/a/m7cqFWJPsWSk8ZnWRN6fR5m>, továbbá Schendrian, 2002
- [5] Pinakin Ariwal: 12 Popular Benefits of RPA in Business, URL: <https://marutitech.com/benefits-of-rpa-in-business/>
- [6] Deloitte: Global RPA Survey, 2018, URL: <https://www2.deloitte.com/content/dam/Deloitte/bg/Documents/technology-media-telecommunications/Deloitte-us-cons-global-rpa-survey.pdf>
- [7] Ian Sommerville: Software Engineering 10th edition (2016), Chapter 2.
- [8] Erik Fogg: Software Development Automation: What Should We Automate?, 2019, URL: <https://prodperfect.com/blog/test-development/automated-software-development/>
- [9] Apache Maven homepage, URL: <https://maven.apache.org/what-is-maven.html>
- [10] Ian Sommerville: Software Engineering 10th edition (2016), Chapter 5.

- [11] Ian Sommerville: Software Engineering 10th edition (2016), Chapter 8.
- [12] Ronald Cummings-John: What is automation testing?, URL:
<https://www.globalapptesting.com/blog/what-is-automation-testing>
- [13] TechTarget: Unit testing definition, URL:
<https://www.techtarget.com/searchsoftwarequality/definition/unit-testing>
- [14] Selenium dokumentáció, URL: <https://www.selenium.dev/documentation/>
- [15] Fekete Antal Kristóf: Kódminőség mérése a programozás tanulását támogató rendszerekben. Multidiszciplináris Tudományok, 2020, 10.2: 469-480., DOI:
<https://doi.org/10.35925/j.multi.2020.2.52> és Fekete Antal Kristóf: Programozást gyakoroltató webes alkalmazás fejlesztése (szakdolgozat)
- [16] Ficsor Lajos, Kovács László, Kusper Gábor, Krizsán Zoltán: Szoftvertesztelés, URL:
https://regi.tankonyvtar.hu/hu/tartalom/tamop425/0046_szoftvertesztelés/adatok.html
- [17] Richard Bellairs, What Is Code Quality? And How to Improve Code Quality, 2019, URL: <https://www.perforce.com/blog/sca/what-code-quality-and-how-improve-code-quality>
- [18] GOPAL, Madhwaraj Kango. Design Quality Metrics on the Package Maintainability and Reliability of Open Source Software. International Journal of Intelligent Engineering and Systems, 2016, 9.4: 195-204.
- [19] ISO/IEC 25010:2011 4.2.7, URL: <https://www.iso.org/obp/ui#iso:std:iso-iec:25010:ed-1:v1:en>
- [20] ISTVÁN, Kun. INFORMATIKAI RENDSZEREK MEGBÍZHATÓSÁGÁNAK MATEMATIKAI MODELLEZÉSE, URL:
http://www.hadmernok.hu/2009_4_kun.pdf
- [21] Measurement of Halstead Metrics with Testwell CMT++ and CMTJava (Complexity Measures Tool), 2017, URL:
https://www.verifysoft.com/en_halstead_metrics.html
- [22] Thomas J. McCabe: A Complexity Measure IEEE Transactions on Software Engineering Vol. 2, No. 4, p. 308 (1976)

- [23] MOONEY, James D. Bringing portability to the software process. Dept. of Statistics and Comp. Sci., West Virginia Univ., Morgantown WV, 1997.
- [24] Dr. Ulbert Zsolt: Szoftverfejlesztési folyamatok és szoftver minőségbiztosítás, 2014, URL: https://regi.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0042_szoftverfejlesztési_folyamatok_magyar
- [25] Ficsor Lajos, Krizsán Zoltán, Mileff Péter: Szoftverfejlesztés, URL: https://regi.tankonyvtar.hu/hu/tartalom/tamop425/0046_szoftverfejlesztés/index.html
- [26] GitHub dokumentáció, URL: <https://docs.github.com/en>
- [27] GitHub Classroom homepage, URL: <https://classroom.github.com/>
- [28] CUnit homepage, URL: <https://cunit.sourceforge.net/>

A megjelölt linkek elérhetőségét utoljára 2022. november 7-én ellenőriztem.