Feladatom: Hibás osztály automatikus keresése hibajelentés (bug report) alapján. Megoldásként a hibabejelentésben szereplő szavak és az osztályokban szereplő azonosító nevek közötti egyezéseket vizsgálom szövegbányászati módszerekkel.

Az általam választott project melyen a program dolgozni fog a bugzilla hibakezelő rendszerben kezelt mozilla nyílt forráskódú verziókezelt projecthez kapcsolódó bejelentések részhalmaza (<https://bugzilla.mozilla.org/>). Ezek a részek java programnyelven készültek. A projekt forráskódja a <https://github.com/mozilla/gecko-dev.git> címen elérhető.

A program fejlesztéséhez a github verziókezelőt is használom többek között a munkám mobilizálása és az adatok biztos tárolása végett.

A program jelenlegi állapota elérhető: https://github.com/krcsmarikrobin/bugassist.git

Az eddig megvalósított részek röviden:

- Első lépés: A GitRepoData.java osztály a JGit API segítségével a letöltött repositoryból kigyűjti a commitokat, ahol a commit üzenetből beazonosítja a bug report azonosítóját, és a committal módosított fájl listát, azon belül is .java kiterjesztésűeket, majd letárolja azokat.

- Második lépés: A HttpBugData.java osztály a JSON API segítéségével a begyűjtött bug report azonosítók alapján sorra lekérdezi a bugzilla.mozilla.org-ról a hibabejelentések adatait úgy, mint a bug rövid és hosszú leírását, a modul nevét és a hibabejelentés állapotát. Ez rendkívül időigényes feladat, mivel kb 9500 java részt érintő bug report van és ezeket egyenként kell lekérdezni http-n keresztül a bugzilláról. Az előzőekre figyelemmel mind az első, mind a második lépés eredményét a JDBC API segítségével a program sqllite adatbázisba menti, és ha a http lekérdezés véletlen megszakad, a program újra sorra veszi az első lépésben kigyűjtött bug azonosítókat ahol még nincs http adat kigyűjtve.

A létrehozott adatbázis sémája:

bugfiles(commitname text, filename text); bughttpdata(bugid integer, shortdesc text, longdesc text, productname text, status text, bagofwords text); bug(commitname text, parentcommitname text, bugid integer);

Az adatbázist a BugDaoSqliteImp.java osztályon keresztül éri el a program.

- Harmadik lépés: a BagOfWords.java osztály elvégzi a gyűjtőnéven dokumentumok előfeldolgozását. Konstruktorai segítségével meghívható hibabejelentésre (melyet a Bug.java bean osztály reprezentál) és meghívható fájlra a forrásfájlok feldolgozása érdekében. A program ezen része a dokumentumokat tokenizálja, kiszűri a stop szavakat, és az így kapott úgynevezett szózsákot (megvalósítva szöveg tömböt) lemmatizálja a ragozások kiszűrése végett. A tokenizálás, stop szó szűrés és a lemmatizálás az Open NLP API segítségével valósul meg. A rendelkezésre álló durván kb 9000 hibabejelentés és durván kb 3000 forrásfájl feldolgozása rendkívül időigényes, ezért az osztály implementálja a Runnable interfészt a szálkezelés megvalósítása érdekében.

- Negyedik lépésben a PreprocessVSM.java osztály a rendelkezésre álló git repositoryban tárolt .java kiterjesztésű forrás fájlok és a hibabejelentések (Bug objektumok) alapján (executor segítségével megvalósított) szálkezelést használva minden hibabejelntéshez és forrásfájlhoz a BagOfWords osztály példányosításával létrehozza a szózsák modelleket a vektortérmodell (vsm) előállításához. Létrehozza továbbá a szózsákokból a vsm modellhez tartozó szótárat is.

- Ötödik lépés: A szótár és a szózsákok segítségével a VsmModel.java osztály előállítja a vsm modellt. Az előállított modell áll egy szótár és egy BagOfWords objektum példányokból előállított kollekcióból, valamint ezek relációját tükröző kétdimenziós tömbből. Itt került megvalósításra még a forrásfájlokból álló BogOfWords objektum példányokból és a hibabejelentésekből álló BogOfWords objektum példányokból álló kollekciók és ezek relációját prezentáló kétdimenziós tömb. Az utóbbi tömbből állapítható meg, hogy melyik hibabejelentéshez mely fájlok javítása volt szükséges.

Jelenleg még a grafikus felültetet egyáltan nem valósítottam meg, a programot a main metódussal szekvenciálisan irányítom, mindig a szükséges részek működését biztosítva. Terveim szerint a következő lépések a programban:

- A vsm modellben kiszámítom az ún. tf-idg súlyozást.

- Minden forrásfájl és hibabejelntés vektora között kiszámítom a cos távolságot (s1).

- Egy forrásfájlt több hibabejelentés is érinthet. Minden forrásfájl és hibabejelentés között kiszámítom az aktuális hibabajelentés és a forrásfálj javítását érintő hibajelentések közötti cos távolságot (s2).

- Minden forrásfájl és hibabajelentés között ha a hibabejelentés tartalmazza az osztály nevét akkor s3= az osztály nevének darabszáma, ha nem tartalmazza akkor s3=0.

- Minden forrásfájl és hibabajelentés között a forrásfájl legutóbbi javításának figyelembevétele érdekében a két hibabejelentés között eltelt hónapok számának reciproka s4 lesz.

- Minden forrásfájl és hibabajelentés között a forrásfájl javításának számának figyelembevétele érdekében s5 egyenlő lesz a bejelentés megtétele előtti javítások számával

Az előzőekben kiszámított s1, s2, s3, s4 és s5 súlyozott összege adja meg, hogy az aktuális hibabejelentéshez mennyire releváns egy forráskód. Rangsorolás lesz hibabajelentésekénta fájlok között.

- A súlyok meghatározásához és a rangsoroláshoz ún. SVM Rank, szupportvektor-gép (SVM) osztályozót fogok használni. Itt még nem döntöttem el, hogy a WEKA API-ját vagy a <https://github.com/ZhouYeJava/RankingSVM> címen elérhető implementált modellt használom vagy mást.

- az eredményeket kiértékelem a kijelölt teszt adatokon.

- Implementálom a grafikus felületet a program könnyed használatához

- Az eredményeket összehasonlítom más hasonló területen folytatott kutatások eredményeivel.

A fentiekben említett súlyozási módszert az alábbi kutatásból merítettem:

<https://www.researchgate.net/publication/263469498_Learning_to_Rank_Relevant_Files_for_Bug_Reports_using_Domain_Knowledge>