EÖTVÖS LORÁND TUDOMÁNYEGYETEM

Informatikai Kar

Komputeralgebra Tanszék

Termelékenységi trendek, minták elemzése szoftverfejlesztési projektekben

Témavezető:

Dr. Kovács Attila

egyetemi docens

Komputeralgebra Tanszék

ELTE IK

Szerző:

Zsiga Attila

programtervező informatikus MSc

nappali tagozat

ELTE IK

Budapest, 2019

Köszönetnyilvánítás

Köszönöm **Dr. Kovács Attila** egyetemi docens támavezetői munkáját és hozzájárulását a diplomamunkámhoz.

Külön köszönet illeti **Szabados Kristófot**, amiért végig segítette a munkámat, az ipari partnerkapcsolaton keresztül adatforrásokkal és háttérinformációkkal szolgált, valamint a belső információk által ellenőrizte a mérések által kapott eredményeimet.

Valamint ezúton szeretnék köszönetet mondani az Ericsson Magyarország Kft.-nek, amiért ipari projektjeik mérésével bővíthettem a kutatásom.

Tartalom

1 Bevezetés 1

1.1 Szoftverprojekt termelékenységének jelentősége 1

1.2 Szoftverprojekt termelékenységének mérési lehetőségei 2

1.3 Termelékenységi mítoszok és viták szakmai körökben 3

1.4 Termelékenység a szakirodalomban 5

1.4.1 Lehman törvények (Lehman's laws of software evolution) 5

1.4.2 Conway törvénye 6

2 Verziókezelő rendszerből termelékenység vizsgálat 8

2.1 Koncepció 8

2.2 Néhány jelentős verziókezelő rendszer 8

2.3 Piper, a Google saját monolitikus verziókezelő rendszere 9

2.3.1 Verziókezelés Google méretekben 9

2.3.2 A commitok alakulása számszerűen 10

2.3.3 A központosított és az elosztott verziókezelő rendszerek dilemmája 12

2.4 Egy népszerű verziókezelő rendszer: Git 12

2.5 Git alapfogalmak 16

3 Téma felmérése, előzetes megfigyelések és az első eredmények kiértékelése 18

3.1 Mérési adatok validálása 18

3.2 Termelékenységbecslés lineáris regresszióval 18

3.3 Témához kapcsolódó projektek 20

3.4 Miért kellett saját megoldás? 20

3.5 Implementációs döntések 21

4 GitRepoAnalyzer program bemutatása 23

4.1 A program célja és feladata 23

4.2 A program felépítése 23

4.3 A program futtatása és a futtatásához szükséges környezet 24

4.4 A program használata 24
4.4.1 Menüpontok és funkciók bemutatása egy általános szcenárión keresztül 25
4.4.2 Kimeneti adatok 30
5 Repositoryk elemzése 37
5.1 A vizsgált repositoryk általános jellemzői 37
5.2 A repository mérések bemutatása 38
5.3 Eredmények 41
5.3.1 Ipari partnerhez kapcsolódó repositoryk 41
5.3.2 Kis és közepes méretű GitHub repositoryk 51
5.3.3 Nagyméretű GitHub repositoryk 82
5.3.4 Osmocom projekthez kapcsolódó repositoryk 88
6 Összefoglalás 93
6.1 A kitűzött cél és a feladat bemutatása 93
6.2 A téma szakirodalmi és gyakorlati felmérése 93
6.3 A vizsgálat kiterjesztése automatizálással 93
6.4 Mérési eredmények értékelése 94
6.5 A célkitűzés áttekintése 94
6.6 További kutatási lehetőségek 94
7 Adathordozó tartalma 95
8 Irodalomjegyzék 96

1 Bevezetés

1.1 Szoftverprojekt termelékenységének jelentősége

Manapság egyre inkább az életünk részévé válnak a szoftverek. A szoftverek mind közvetlenül, mind közvetve hatással vannak a mindennapi életünkre. Fontos alapjait jelentik többek között a kereskedelemnek, a szolgáltatásoknak, az energiagazdálkodásnak és ezáltal az emberek életét befolyásoló gazdaságnak. Mindemellett szociális oldalról a mai kommunikáció kulcstényezői, az orvostudomány és ezáltal az egészségügy fejlődésének elősegítői.

Számos ilyen fontos és nagy kihatással bíró rendszernél már nem csak az számít, hogy funkcionálisan mit tud, hanem az is, hogy mikor válik elérhetővé. A gyorsaságnak nem csak pénzügyi haszna lehet, mindenképpen üzleti előnyt is jelent, ha egy szoftver termék a konkurenciánál hamarabb kerül a piacra. Tekintsük példaként a jelenleg nagy érdeklődést kiváltó önvezető autókat. Érdekes belegondolni abba, hogy a WHO szerint az utóbbi pár évet tekintve évente nagyságrendileg 1,2 millió ember hunyt el az utakon halálos kimenetelű közúti balesetben.

- A Global status report on road safety 2013-as kiadása szerint éves szinten 1,24 millió ember hal meg közúti balesetekben. [1] [2]
- A Global status report on road safety 2015-ös kiadása szerint éves szinten 1,25 millió ember hal meg közúti balesetekben. [3] [4]

Ezek a számok átlagosan napi szinten több, mint 3 000 ember halálát jelentik.

Feltéve, hogy előbb-utóbb sikerül kifejleszteni egy olyan önvezető járműtechnológiát, amely az emberi tényezőnél nagyobb biztonságot jelent a közlekedésben, ezek a számok akár ugrásszerűen is javulhatnak. Ilyen irányú projekteket már is fejlesztenek, sőt a kontrolált körülmények közötti tesztelés fázisánál tartanak. Azonban nem csak az a kérdés, hogy megoldható-e, az is fontos, hogy mikorra. A példa esetében minden egyes nap, ami nem viszi előre ezen projektek fejlődését, fejlesztését, statisztikailag több mint 3 000, a közlekedésbiztonságot növelő szoftverek által egyébként megmenthető ember halálát jelentheti. Tehát az, hogy csupán egyetlen nappal hamarabb lesz kiadható és elérhető egy közlekedésbiztonsági szoftver, akár 3 000 ember életét is megmentheti.

A fent bemutatott példából egyértelmű, hogy miért kiemelten fontos a szoftverek fejlesztésénél a termelékenység vizsgálata, becslése és optimalizálása. Fontos tisztában

lenni azzal, hogy milyen tényezők és döntések milyen irányba befolyásolják a termelékenységet és hatnak ki a projekt fejlődési ütemére. Fel kell ismerni a fejlesztés sebességét befolyásoló, gyorsító és lassító tényezőket.

1.2 Szoftverprojekt termelékenységének mérési lehetőségei

A szoftverprojektek termelékenységének vizsgálatát több irányból is meg lehet közelíteni. Például fel lehet mérni a projektben résztvevők észrevételeit, meglátásait, a számukra problémásabb vagy nagyobb stressz szinttel járó feladatokat, melyeket mind emberi, mind projekt előrehaladási szempontból előnyös orvosolni. Meg lehetne vizsgálni a fejlesztésekhez kapcsolódó statisztikákat, diagramokat, melyekből látszódna, hogy mennyi hiba és fejlesztési feladat készül el átlagosan egy iterációs időszak alatt. Bizonyára sok esetben a projekt és azon belül a különböző részlegek, csapatok vezetői is tudnának hasznos információkkal szolgálni arról, hogy milyen módszerekkel, módszertanokkal dolgoztak és milyen tapasztalatokat láttak, melyik milyen előnyöket és hátrányokat jelentett. Az ilyen irányú kutatáshoz azonban, egyesével minden projektnél el kell érni a felmérésben való részvételi hajlandóságot. Ráadásul nem csak a vezetői, hanem a fejlesztésben résztvevők közötti minél szélesebb körben. Ha ez sikerül is, akkor sem biztos, hogy a résztvevők által fontosnak érzett tényezők a valóságban is annyira fontosak.

Van azonban egy másik vizsgálati lehetőség is, melyet a verziókezelő rendszerek kínálnak. Ha elérhető, például nyíltforráskódú projektről beszélünk, akkor úgy is nyerhetünk ki használható eredményeket, hogy nem az előzőekben említett emberi és szervezeti információk alapján, hanem az adott szoftver forráskódjának változása alapján teszünk megfigyeléseket. A verziókezelő rendszerek által tárolt információ rendelkezésre áll, könnyen elérhető és jól mérhető. Ráadásul kellően tényszerű, precízen mérhető és kizárja az esetlegesen eredményeket torzító emberi tényezőt.

Minél nagyobb és változatosabb a vizsgált projekthalmaz, vélhetően annál közelebb állhatnak az ilyen irányú empirikus megfigyelések eredményei a valósághoz. A munkám során tehát empirikus megfigyeléseket végeztem egy szoftverprojekthalmaz verziókezelő rendszereiből kinyerhető előzmény információi alapján.

A teljesség igényével itt érdemes megjegyezni, hogy a commitok és a módosított sorok számának alakulásán alapuló vizsgálat nem feltétlenül objektív mérési módszere a termelékenységnek. A termék fejlődésének, termelékenységének vizsgálatakor

különböző szempontok alapján szubjektív vélemények, konklúziók alakulnak ki. Az egyik legtriviálisabban felmerülő kérdéses szempont, hogy egyenértékűnek tekinthető-e minden commit vagy módosított sor. Legyen szó akár egy architektúrális, teljes termékre kihatással lévő kódsorról, vagy „csak” egy fejlesztőknek szóló és a termék működését nem befolyásoló komment sorról. Nézőpont és döntés kérdése, hogy egyenértékűnek tekinthető-e vagy sem. A jelen mérési módszer az ilyesfajta szubjektív súlyozástól mentes, ugyanakkor épp ilyen okból lehetséges, hogy más termelékenységi mutatók figyelembevételéhez képest a jelen mérési módszer és annak eredményei független vagy irreleváns eseményeket jelentenek.

Egy másik tényező, ami torzíthatja az eredményeket az, ha az adatokat előállító személyek esetleg érdekeltek azok torzításában. Ilyen legkönnyebben akkor képzelhető el, ha a fejlesztők fizetése vagy azon felüli juttatásainak mértéke esetleg függ az adott időszak alatt megírt kódsorok számától. Mivel a méréseimet számos évvel a konkrét fejlesztések után, és azoktól teljesen függetlenül végeztem, így ilyen jellegű közvetlen hatás kizárható. Azonban, ha a megfigyelt projektek valamelyikében volt ilyen (akár időszakos, akár hosszútávú) jutalmazási rendszer, az abból adódó torzulások az én méréseimben is megjelenhetnek. Ez ellen úgy probálok védekezni, hogy számos különböző projektet vizsgálok meg (különböző programozási nyelvben, fejlesztési közegben és céllal fejlesztetteket).

1.3 Termelékenységi mítoszok és viták szakmai körökben

A következőkben sorra veszek néhány programozói körökben elterjedt gondolatot. Jelen esetben ezeket a gondolatokat nevezhetnénk mítosznak is, mert jellemzően az adott témával kapcsolatban nincsenek egyértelműen, tudományosan bizonyított teljeskörű ismeretek. Sok esetben csupán arról van szó, hogy valamikor felmerült egy kellő bizonyíthatóság hiányában eldönthetetlen igazság tartalmú állítás, és arról időről-időre, két táborra szakadva, viták folynak a programozók között. Más esetekben pedig, bár teljesen logikusnak tűnnek a következtetések, még akár mérések is lehetnek 1-1 részhatás szemszögéből, de nem egyértelmű, hogy ezek a teljes képen milyen erősen jelennek meg vagy megjelennek-e egyáltalán.

Nézzünk meg pár ilyen mítoszt:

- A „The 10x developer” feltevés szerint léteznek olyan fejlesztők, akik 10-szer hatékonyabbak, mint a többi fejlesztő. [5] [6] [7]

- A programnyelvek témakörében gyakran felmerülő állítás például, hogy C++ nyelven nagyságrendekkel gyorsabban lehet fejleszteni, mint Assemblyben. Sokkal jobb minőségű, sokkal jobb teljesítményű programot lehet készíteni. Ehhez hasonló állítással egyébként szinte bármely két programozási nyelv esetén találkozhatunk. Legyen szó akár egymáshoz közelálló vagy hasonló nyelvekről, vagy két teljesen más felépítésű, működésű és célú nyelvről. [8]
- A szoftverfejlesztési módszertanok esetében gyakran és büszkén hangoztatják, hogy éppen aktuálisan melyik az a módszertan (lásd: Scrum, Kanban, DSDM, XP, ICE, DevOps, LEAN, RUP, stb…) és gyakorlat (lásd: TDD, BDD, DDD, CI, CD), mellyel minden szoftverprojekt termelékenysége ugrásszerűen nőni fog. Aktuálisan ilyenek az agilis módszertanok, melyek bevezetése után általában, minden egyéb körülmény megváltoztatása nélkül is többszörös termelékenységnövekedést ígérnek.
- Sokszor merül fel, hogy a boldog programozó jobb és több kódot ír. Ezért érdemes a munkahelyen szórakozási lehetőségeket biztosítani a számukra (lásd: Csocsó asztal, szabadon használható játékkonzol, stb…).
- Az is teljesen logikusnak hangzik, hogy akár csak egyetlen programozó távozása is jelentős hátrányt jelent a cégnek. Hiszen csökken a termelés, és ha tudják is pótolni a kieső személyt, az új fejlesztő lassan tanul be, sőt a betanítása egy másik szoftverfejlesztő erőforrásait is leköti. Így akár többszörös kár is érheti a projektet.
- Gyakran hangzik el az az állítás is, hogy a szoftverfejlesztés olyan kreatív gondolkodást igénylő munka, ami mindenféleképpen nyugodt munkakörülményeket kíván meg. Viszont azt is látni, hogy a nagy zajjal járó nyitott iroda jellegű munkakörnyezet egyre inkább terjedőben van.
- Érdekes módon ketté tud szakadni a fizetések mentén is a mítoszok halmaza. Az egyik oldalon azok állnak, akik természetesnek veszik, hogy a jó programozókat jól meg is kell(-ene) fizetni. A másik oldalon viszont arra hívják fel a figyelmet, hogy a startupoktól a legnagyobb cégekig az igazán nagy „innovációk” mindig a legnagyobb megszorítások idején jelentek meg. (Lásd, kollégiumban megírt Facebook, pizzán élő rockstar startupper programozók, diplomamunkaként elkészült Google kereső, stb…)

Feltéve, hogy az ilyen közhiedelmek nem alaptalanok, még ha nem is feltétlenül tudományosan bizonyítottak vagy bizonyíthatók, akkor ezekből következően egy kellően

nagy véletlenszerűen kiválasztott szoftverprojekthalmaz megvizsgálásakor várhatóan láthatóak lesznek a felvetésekben állítottak.

1.4 Termelékenység a szakirodalomban

A szoftverprojektek termelékenységének vizsgálatakor érdemes megemlíteni az alábbi szakirodalmi publikációkat.

1.4.1 Lehman törvények (Lehman's laws of software evolution)

A szoftver evolúcióval foglalkozó publikációk közül az egyik legszélesebb körben elfogadott tanulmány a Meir Lehman által először 1980-ban publikált „*Programs, Life Cycles, and Laws of Software Evolution*", amelyre csak, mint Lehman törvények szoktak hivatkozni. Ezekben a szoftver evolúciós folyamatára vonatkozó általános érvényű elveket fogalmaznak meg. Leegyszerűsítve az alábbi pontokban foglalhatók össze a törvények, melyek közül a jelen munkához kapcsolódó részek vastag betűvel lettek kiemelve: [9] [10] [11]

- „Continuing Change": A szoftvernek folyamatosan igazodnia kell az igényekhez, máskülönben egyre kevésbé lesz kielégítő a működése.
- „Increasing Complexity": A szoftver fejlődése növeli annak komplexitását, ha az nincs karban tartva vagy egyszerűsítve.
- „Self Regulation": A szoftver evolúciós folyamata önszabályozó, a mérőszámai normál eloszláshoz közeliek.
- „Conservation of Organisational Stability (invariant work rate)": **A szoftver átlagos aktív fejlődésének mértéke közel állandó a teljes életciklusa alatt.**
- „Conservation of Familiarity": **A szoftver fejlesztésének minden résztvevője aktívan kiveszi a részét a projektből. Az emberi erőforrás túlzott növelése azonban mérsékeli ezt. Ezért az átlagos inkrementális növekedés invariáns marad a szoftver fejlődése során.**
- „Continuing Growth": A szoftver funkcionális tartalmának folyamatosan nőnie kell, hogy fenntartsa az elégedettséget a teljes életciklus folyamán.
- „Declining Quality": A szoftver egyre kevésbé fog megfelelni a minőségi elvárásoknak, ha nem tartják szigorúan karban és alkalmazkodnak az operatív környezeti változásokhoz.
- „Feedback System": A szoftver evolúciós folyamata során a jelentős javulás elérése érdekében figyelembe kell venni a visszajelzéseket.

1.4.2 Conway törvénye

Melvin Edward Conway programozó 1967-ben előállt egy megfigyeléssel, melyre 1968-tól már Conway törvényeként hivatkoznak. Ennek az alapvető tézise a következő:

„*organizations which design systems… are constrained to produce designs which are copies of the communication structures of these organizations.*” [12] [13]

Vagyis azok a szervezetek, amelyek rendszereket terveznek, arra vannak korlátozva, hogy olyan terveket készítsenek, amelyek a saját kommunikációs struktúrájuk másolata.

A szoftverfejlesztésre vetítve a megfigyelés lényege úgy érzékeltethető, hogy ahhoz, hogy egy szoftver két modulja korrekten együttműködjön, azok tervezőinek és fejlesztőinek kommunikálniuk kell. Így a szoftver struktúrája követni fogja a szervezet struktúráját.

Ebből adódóan, bár lehetséges, hogy a szoftver egyes önálló részei valóban főleg a lokális technikai tényezőktől függenek, de ha komplex rendszerként nézzük meg a teljes szoftverterméket, akkor a különálló részek közti kapcsolatokra, azok működésére és működőképességére sokkal nagyobb hatással vannak a szervezeti egységek közötti erőviszonyok. [14] [15]

Ennek egy kibővitett változata a tükörkép (mirroring) hatás. Eric Steven Raymond amerikai programozó szintén a Conway megállapításához hasonló következtetésekre jutott a „*The New Hacker's Dictionary*” [16] című könyvében. Ebben arról ír, hogy a szoftver struktúrája egybevágó lesz a fejlesztő csapat struktúrájával. Nem csak fejlesztői, hanem fejlesztői csapat szinten is felfedezhetők minták a megírt kódsorokban. Tehát nem csak egyénre, hanem csapatra jellemző sajátosságok is megfigyelhetők.

James O. Coplien és Neil B. Harrison az „*Organizational Patterns of Agile Software Development*” [17] című munkája szerint egy projekt, s ezától a fejlesztő cég számára komoly problémát jelenthet, ha a cég szervezeti felépítése és kapcsolatai nem tükrözik a termék alapvető részeit és a köztük lévő kapcsolatokat. Vagyis a fejlesztendő termék architektúrájának és a fejlesztő szervezet felépítésének összeegyeztethetőnek kell lennie.

Conway törvényével több ízben is foglalkoztak kutatások és tanulmányok. Ilyen például Alan MacCormack, John Rusnak és Carliss Baldwin „*Exploring the Duality between Product and Organizational Architectures: A Test of the “Mirroring” Hypothesis*” [18]

című munkája, melynek eredménye megerősíti a tükörkép feltételezést. Tanulmányukban jelentős különbségeket fedeztek fel a földrajzilag egymástól távoleső fejlesztőcsapatok által fejlesztett szoftverek modularitásában.

Illetve a Microsoft Research is közölt empirikus adatokra épülő esettanulmányt Conway törvényével kapcsolatban. Nachiappan Nagappan, Brendan Murphy és Victor R. Basili „*The Influence of Organizational Structure on Software Quality: An Empirical Case Study*" [19] névvel megjelent esettanulmány empirikus vizsgálattal kimutatta, hogy a termék minőségét erősen befolyásolja a szervezet struktúrája. Mivel egy új szoftver megírásához nem feltétlenül jön létre egy új szervezet, ezért az elkészítendő szoftver architektúrális tényezői már azelőtt eldőlhetnek, hogy az adott szoftver igénye megtalálja az adott (már meglévő) fejlesztői szervezetet, tehát még az igények és követelmények felmérése és a specifikáció előtt. Másrészt ez azt is jelenti, hogy szükség esetén azon csak komoly szervezeti átalakítások révén lehet lényegesen változtatni.

Kovács Attila és Szabados Kristóf „*Internal quality evolution of a large testsystem–an industrial study*" [20] című publikációjában empirikus megfigyeléseket végeztek két nagyméretű, ipari fejlesztésű tesztrendszer fejlődéstörténetén. A kutatás során tendenciákat és összefüggéseket kerestek a projektek fejlődése, valamint annak körülményei és az úgynevezett Code Smells kódrészletek száma között. A Code Smells kifejezés Kent Beck és Martin Fowler által lett elterjedt, és olyan kódrészleteket jelöl, amelyek önmagukban technikailag nem helytelenek és az adott program vagy funkció működésére nézve sem hibásak, ugyanakkor olyan architekturális vagy hosszútávú problémákat jelezhetnek, amelyekre nagyon nehéz fényt deríteni. A kutatásuk során a tesztrendszereket úgy tekintették, mint tesztelési feladatot ellátó szoftver termék. A projektek öt éves fejlesztési időszakához tartozó adatok és ismeretek által nyomon követték a projektek fejlődését és azok kódminőségének változását. A megfigyelések azt mutatták, hogy sem a Continuous Integration módszer bevezetése, sem a kódminőség javulását segítő eszközök megléte, sem a fejlesztési módszertan váltás (vízesés modellről agilis módszertanra), sem a szervezeti és személyi változások nem okoztak mérhető változást a megfigyelt tendenciákban. A vizsgálat alapján a minőségfejlesztések elsősorban egyéni belső motiváció alapján kerültek megvalósításra. Arra a megállapításra jutottak, hogy a megfigyelt tesztrendszerek előrevetíthető tendenciákat követnek, a Lehman törvényeknek megfelelően. A vizsgált tesztrendszerek fejlődésén tapasztalt megfigyelések hasonlóságot mutattak a szoftverrendszerek fejlődésével.

2 Verziókezelő rendszerből termelékenység vizsgálat

2.1 Koncepció

Manapság minden, több fejlesztőt igénylő szoftverfejlesztési projekt alapvető eleme valamilyen verziókezelő rendszer, amely lehetővé teszi a forráskódon végrehajtott módosítások követhetőségét, tárolását, visszaállíthatóságát, a fejlesztők és különböző fájlverzióik közötti szinkronizációt és összehasonlíthatóságot. Ezáltal a szoftverprojekt fejlődését, változását (legalábbis a programkódot tekintve) jól leírják a projekt fejlesztésénél használt verziókezelő rendszerben tárolt előzmények. Az előzmények ismeretében nem csak a projekt aktuális állapotát elemezhetjük, hanem a korábbi előzmények ismeretében képet kaphatunk bizonyos projektre jellemző múltbéli adatokról is. A legrégebbi, kezdeti információtól indulva, napról-napra végig követhető a forráskód változása, mely jó alapot biztosít a jelenlegi szempontok vizsgálatához.

2.2 Néhány jelentős verziókezelő rendszer

A következő rész egy nagyobb lélegzetvételű idézet lesz, melyre a verziókezelő rendszerek történeti és fejlődési hátterének megismerésekor találtam. [21] Mind tartalmilag, mind megfogalmazásilag megfelelőnek láttam felhasználni. Épp ezért az alábbi részeket, ugyan az eredeti konkextusából kiragadva, de szöveghűen emeltem át.

„Az egyik legkorábbi verziókezelő rendszer a CVS (Concurrent Versions System). Az alapvető igényeket kielégíti, hiszen a fejlesztők lemásolhatják a repositoryt, és a helyi változtatásaikat vissza is írhatják abba. A CVS az ütközéseket eredetileg egy meglehetősen egyszerű, first come, first serve modellel oldotta fel, ami azt jelenti, hogy változtatást mindig csak a szoftver legfrissebb verzióján lehet végrehajtani. Emiatt gyorsan és gyakran kellett a változtatásainkat visszatölteni a repositoryba, nehogy valaki megelőzzön minket. A CVS ma már támogatja a branchinget, így valamivel rugalmasabb. Fő előnye, hogy régóta jelen van, így stabilnak mondható, azonban néhány alapvető hibája miatt (nincsenek atomi műveletek, ezáltal a branching nagyon költséges művelet) ma már kevésbé népszerű.

Az Apache Subversion (SVN) annak idején a CVS alternatívájaként jött létre, ami számos bug kijavítása mellett megpróbált maximális kompatibilitást biztosítani az eredeti rendszerrel. Az adatbázis konzisztenciáját az SVN atomi műveletekkel biztosítja: egy adott változtatásnak vagy minden összetevője bekerül a repositoryba, vagy egy sem. Míg a CVS-ben a branching lassú művelet, és nem is igazán támogatta a rendszer a hosszú

életű brancheket, az SVN-t sokkal inkább erre találták ki. Fő negatívumaként azt szokták felhozni, hogy centralizált: a repository egyetlen központi helyen van tárolva. Ennek az a fő hátránya, hogy ha ez a központi szerver valamiért épp nem elérhető, a fejlesztők nem tudnak hozzáférni a kódbázishoz.

Ennek a problémának az orvosolására hozta létre Linus Torvalds a Git nevű verziókezelő rendszert, amely ma egyértelműen az egyik legnépszerűbbnek számít. A Git egy radikálisan új modellel állt elő a CVS-hez és az SVN-hez képest: a rendszer teljesen elosztott, így nincs szükség egyetlen központi repositoryra. A Git általában véve elődeinél sokkal gyorsabb, és a teljes verziótörténet offline is elérhető minden egyes kliensen.

Említésre méltó versenyző még a Mercurial, amely szintén elosztott, de több hasonlóságot mutat az SVN-el, így talán némileg gyorsabban tanulható azok számára, akik SVN-ről térnek át egy elosztott rendszerre." [21]

2.3 Piper, a Google saját monolitikus verziókezelő rendszere

A Google alkalmazottai 1999 óta egy közös (monolitikus), megosztott kódbázisban dolgoznak, egy centralizált verziókezelő rendszeren keresztül. A Google által fejlesztett szoftverek többsége tehát jelenleg is egy nagy, megosztott repositoryban tárolódik. Az évek alatt azonban a Google szoftverfejlesztőnek száma többszörösére nőtt. A kódbázis mérete pedig exponenciális növekedést mutat. Ez pedig olyan kihívások elé állította a használt verziókezelő rendszert, hogy annak jelentős technológiai fejlődésen kellett átmennie. [22] [23]

2.3.1 Verziókezelés Google méretekben

A Google monolitikus repositoryja egy hatalmas méretűre felskálázott rendszert jelent. Egyben működő példája annak, hogy a teljes kódbázis egyetlen repositoryban való kezelésének modellje sikeresen skálázható.

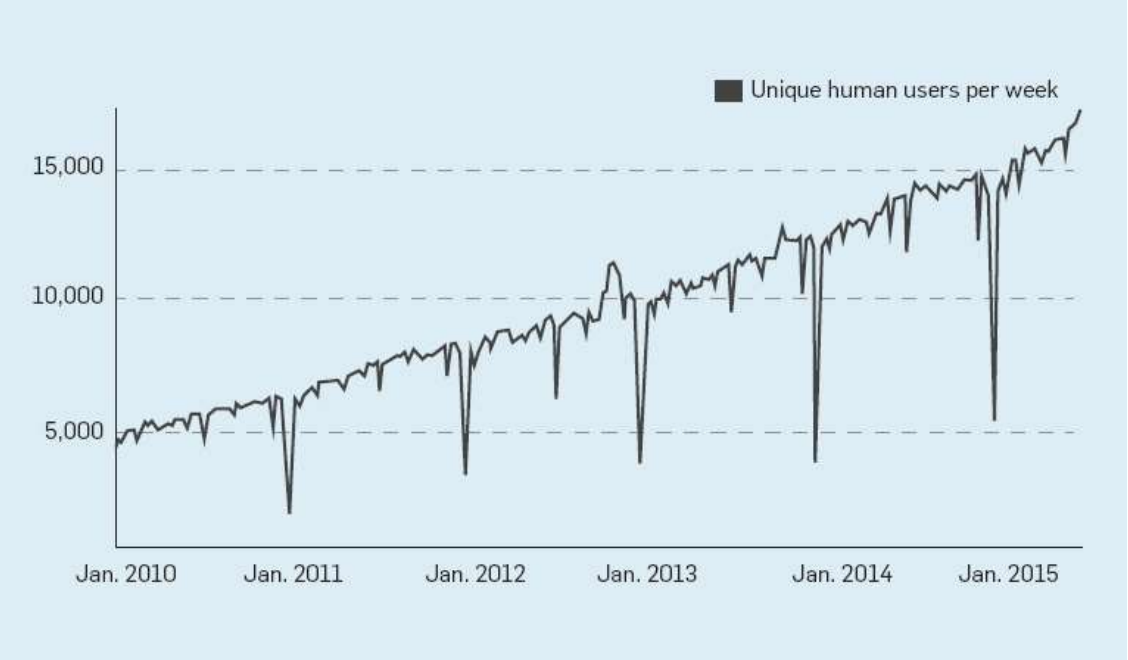
A Google kódbázisa nagyságrendileg egymilliárd fájlt és megközelítőleg 35 millió commitot tartalmaz. Melynek része a megközelítőleg kétmilliárd sornyi kód, 9 millió különböző forrásfájlban. A hatalmas méretű adatmennyiség miatt a repository 86 TB-nyi adatot tartalmaz.

A Google repositoryjában 2014-ben hetente megközelítőleg 15 millió sornyi kód változott megközelítőleg 250 000 fájlban. Összehasonlításképpen a Linux kernel repositoryja összesen megközelítőleg 15 millió sort tartalmaz közel 40 000 fájlban.

A Google kódbázisát több, mint 25 000 fejlesztő használja szerte a világban, akik átlagosan napi 16 000 változást commitolnak. Emellett további napi 24 000 commit keletkezik automatizált rendszerek által. A repository milliárdnyi fájl olvasási kérés mellett csúcsidőben megközelítőleg 800 000 lekérdezést, a teljes napot figyelembevéve átlagosan 500 000 lekérdezést szolgál ki másodpercenként minden egyes munkanapon. Az említett forgalom jelentős részét a Google folyamataihoz tartozó belső rendszerek bonyolítják le.

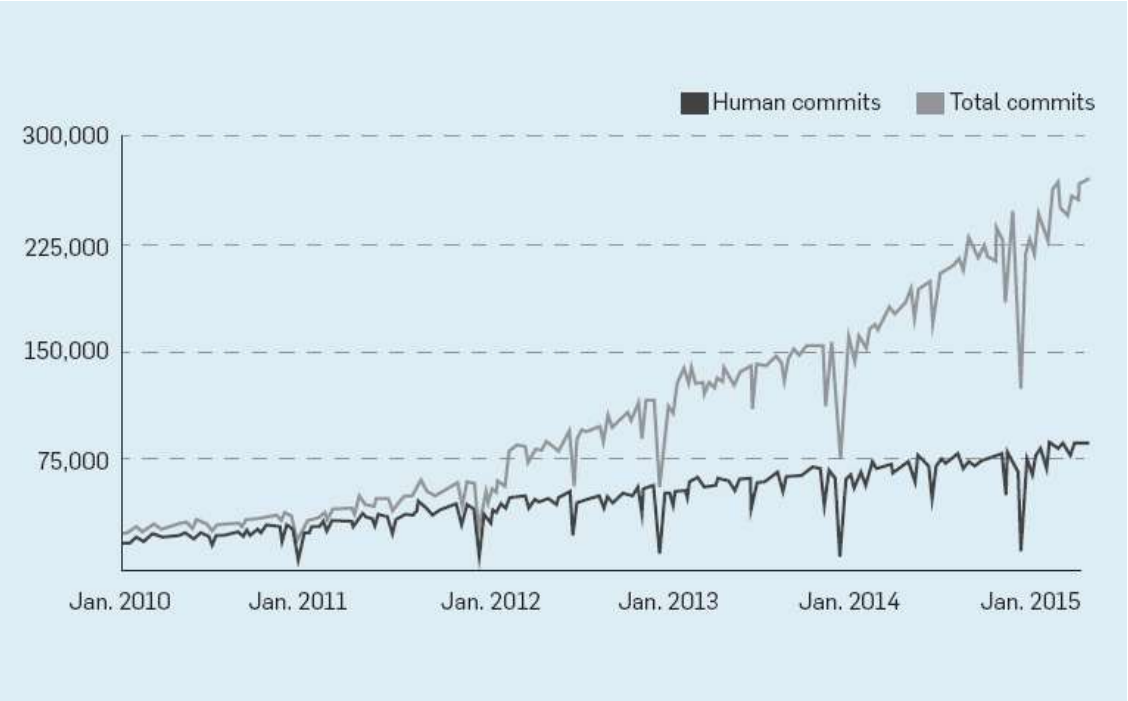
2.3.2 A commitok alakulása számszerűen

Az elérhető adatok azt mutatják, hogy az emberi commitolók száma megközelítőleg lineáris arányban nőtt, ha eltekintünk a minden év végén jelentkező visszaesésektől, amiket az ünnepi időszakok magyarázhatnak.



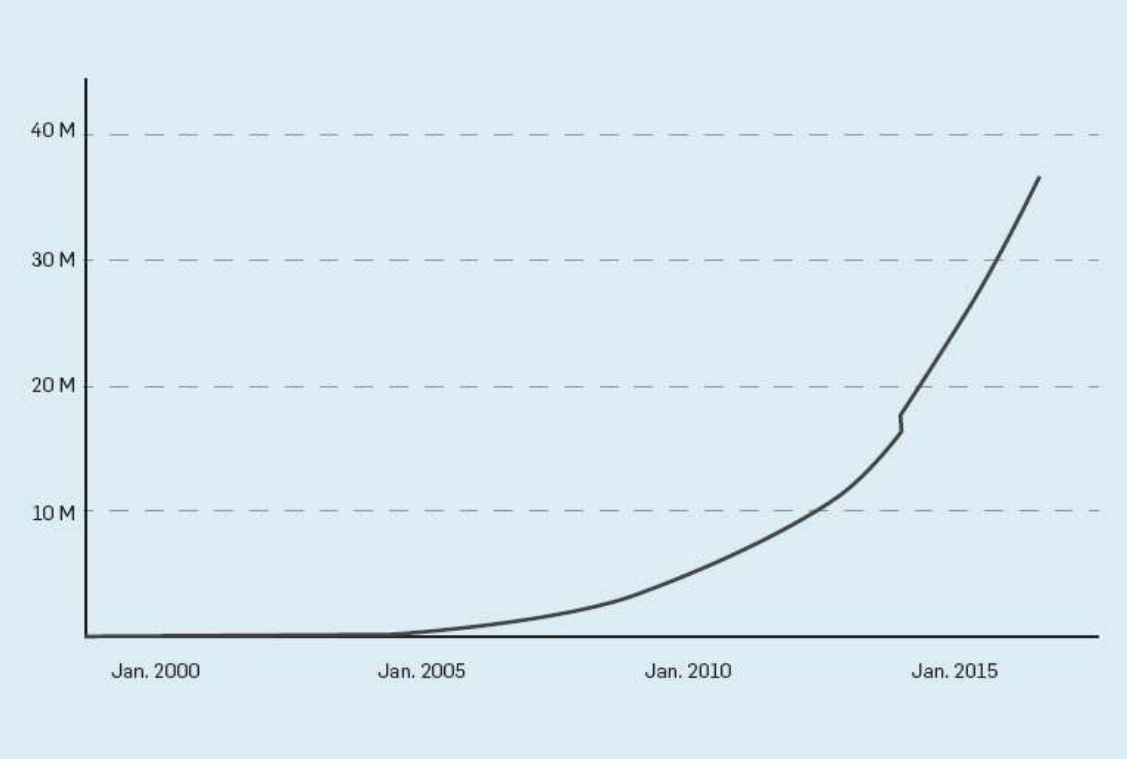
2-1. kép: Commitoló felhasználók száma hetente

A repositoryt használó emberek száma tehát megközelítőleg lineárisan nő. A commitok számának vizsgálatánál azonban meg kell különböztetni két esetet. Az emberek által létrejött commitok száma nagyon hasonló a commitolók számának alakulásához, szintén megközelítőleg lineárisan nőtt. Ismételten azáltal, hogy eltekintünk az év végi, ünnepi időszak körüli visszaesésektől. Azonban, ha az összes commit számát tekintjük, akkor exponenciális növekedés látható.



2-2. kép: Commitok száma hetente

Egészen 2012-ig a commitok meghatározó arányban emberektől származtak. Majd 2012-től az automatizálási törekvéseknek köszönhetően egyre nagyobb arányban generálódnak a commitok. Ezek alapján tehát látszik, hogy a commitok száma egyre nagyobb arányban nem embertől származik, hanem valamilyen automatizmus által létrehozott gépi commit.



2-3. kép: A Google repositoryjába commitolt változások száma milliókban mérve

A Google felhasználási módjára, igényeire és a kódbázisának méretére egyik elérhető verziókezelő rendszer sem tudott megfelelő megoldást nyújtani. Ezért Piper néven saját verziókezelő rendszert készítettek, amely képes megfelelő módon kezelni a hatalmas mennyiségű adatot.

2.3.3 A központosított és az elosztott verziókezelő rendszerek dilemmája

Mivel a kódbázis központosított verziókezelése csak nagy anyagi befektetés árán skálázható, ezért a Google vezetői bizonyos időközönként megvizsgálják, hogy érdemes lenne-e leváltani a monolitikus verziókezelő modellt.

A modell váltáson túl, az elosztott verziókezelő rendszerek egyre nagyobb népszerűsége miatt, időről-időre megvizsgálják, hogy milyen előnyökkel és hátrányokkal járna, ha a saját Piper nevű verziókezelő megoldásukat leváltanák Gitre. Ezt elősegítendő a Googlenél egy külön csapat foglalkozik a Git támogatásával, fejlesztésével. Valamint az Android és a Chrome csapatai a központi repositorytól független Git repositorykat is használnak. A Google Git alapon tárolt Android kódbázisa több, mint 800 különálló repositoryba van szétosztva.

A Git azonban több szempontból sem felel meg a Google és a Piper jelenlegi működésének, sőt akár kifejezetten ellentétes felhasználási módot vár el. Például a Git esetén erősen javasolt több, kisebb méretű repositoryban tárolni a forráskódot, valamint a Git esetén a clone paranccsal az egész repository tartalma egyben rámásolódna a lokális számítógépre. Először is a megfelelő teljesítmény és kezelhetőség elérése érdekében a monolitikus repositoryt több ezer kisebb darabra kéne szétosztani. Ez az átalakítás önmagában komoly belső kulturális és munkafolyamat szintű változást tenne szükségessé. Továbbá ez esetben le kéne mondani a jelenlegi működés előnyeiről, valamint a már meglévő, Piperre kialakított folyamat- és fejlesztéstámogató eszközökről.

Mindent összevetve tehát a Google számára az áttérés egy elosztott verziókezelő rendszerre jelenleg még több hátránnyal járna, mint amennyi előnnyel.

2.4 Egy népszerű verziókezelő rendszer: Git

Az elmúlt pár év felmérései és elemzései alapján megfigyelhető volt egyfajta trend, ahogy a Git nem is olyan lassan felzárkózott a leggyakrabban használt verziókezelő rendszerek közé. Majd többek között a GitHub, a Bitbucket és a saját fejlesztésű Team Foundation Version Control (TFVC) rendszerét alapértelmezetten Gitre váltó Microsoft segítségével

mára az egyik, ha nem a legnépszerűbb választás. Nézzünk pár adatot az elmúlt néhány évből időrendben ennek alátámasztására.

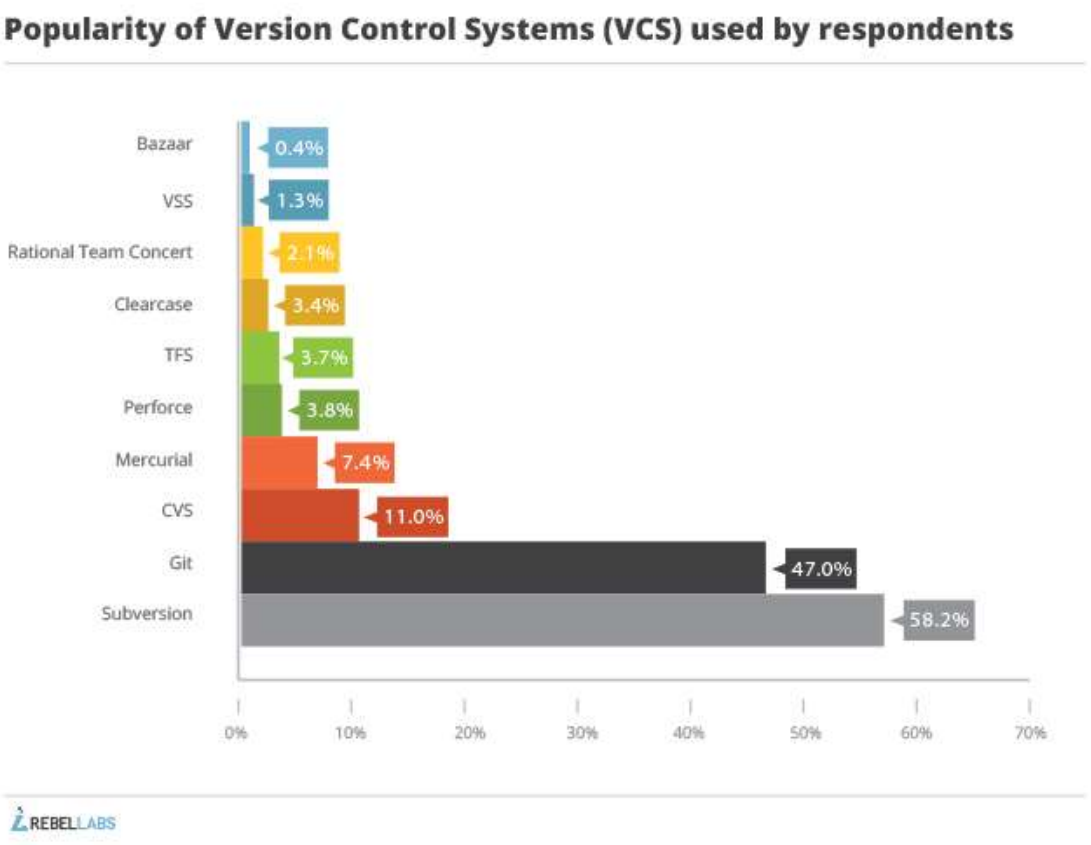
Először két felmérésen alapuló eredmény látható:

- RebelLabs Developer Productivity Report (2012-es és 2013-as adatok) [24]

A RebelLabs 2012-es felmérése alapján a verziókezelők közötti népszerűség az alábbiak szerint alakult:

1. Subversion (66%)
2. Git (33%)
3. CVS (12%)
4. Mercurial (10%)

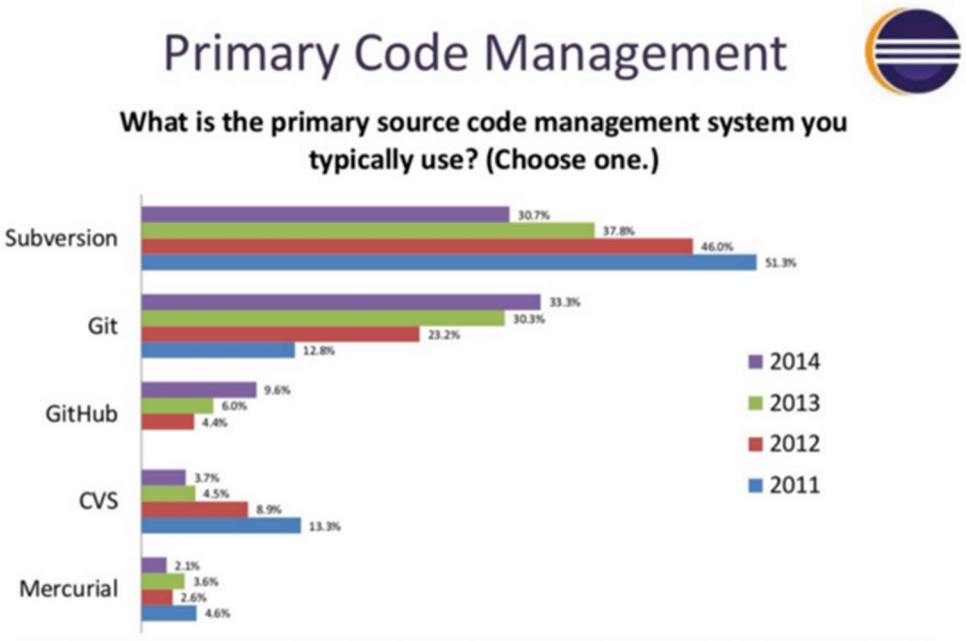
Majd egy évvel később, a 2013-as felmérés az alábbi eredménnyel zárult:



2-4. kép: RebelLabs Developer Productivity Report 2013

Mindössze egy év alatt tehát 14%-ot javult a RebelLabs felmérésének keretében válaszolók szerint a Git népszerűsége, míg az akkori legnépszerűbb Subversion előnye közel 8%-kal csökkent.

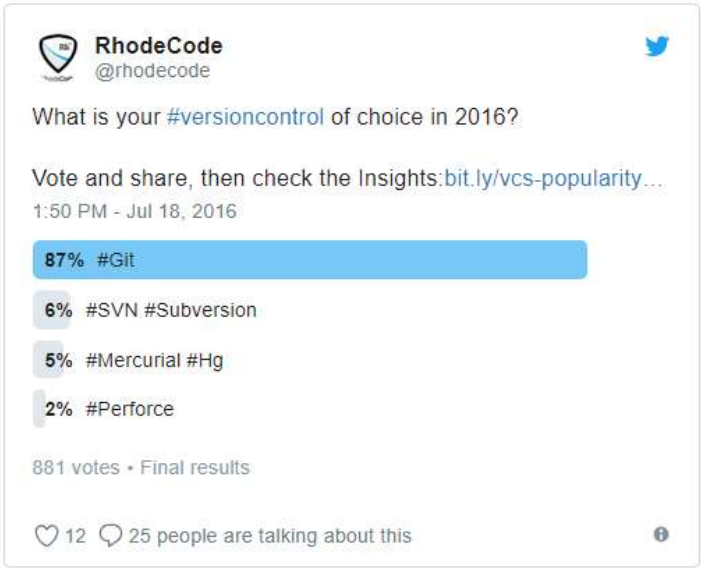
- Eclipse community survey (2014-es adatok) [25]



2-5. kép: Eclipse community survey 2014

Az Eclipse felhasználók között a 2011 és 2014 közötti időszakban nézve évről évre egyre nagyobb arányban választották elsődleges verziókezelő rendszernek a Gitet, illetve a szintén Git alapú GitHubot. Majd 2014-re megelőzte az addigi éllovas Subversiont.

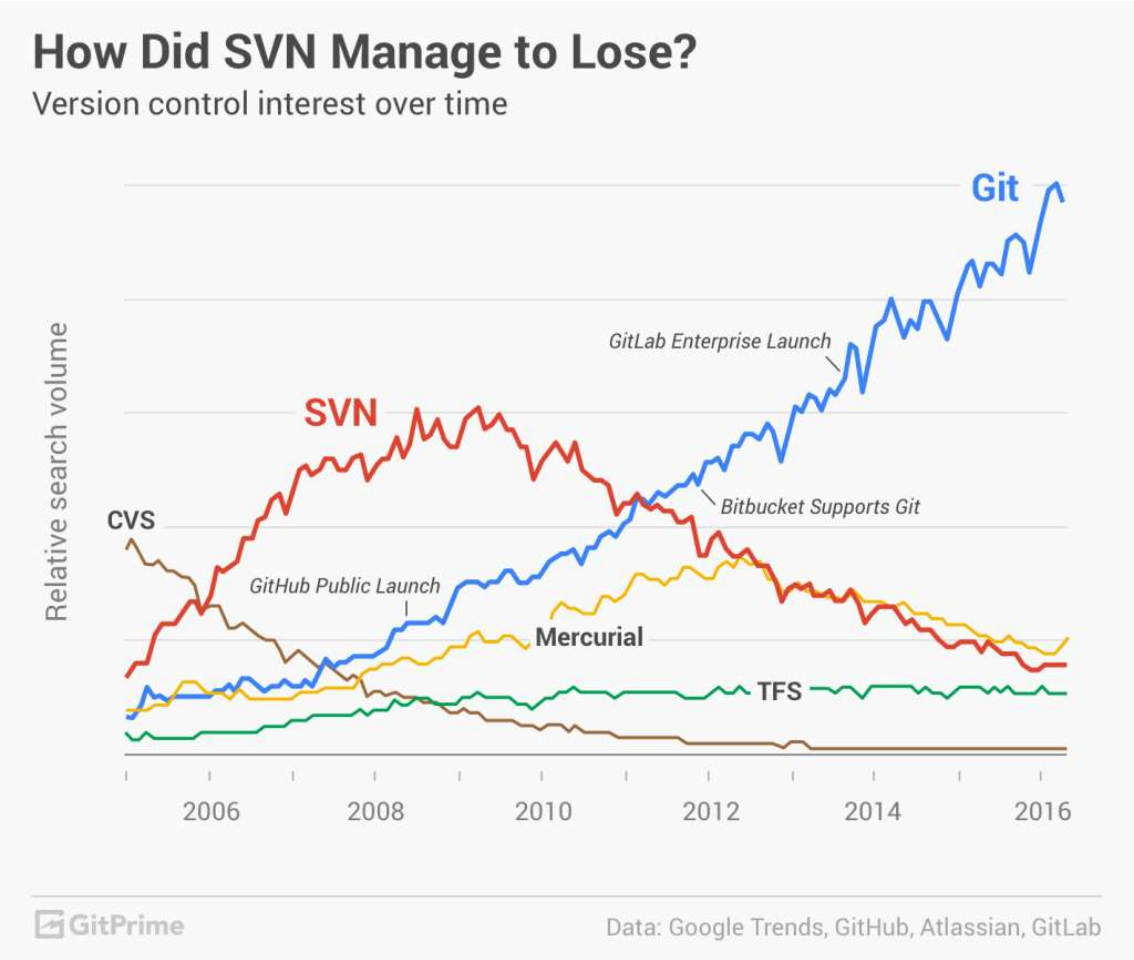
- RhodeCode Twitter szavazása „What is your #versioncontrol of choice in 2016?” címmel (2016-os adatok) [26]



2-6. kép: RhodeCode „What is your #versioncontrol of choice in 2016?”

A RhodeCode 2016-os Twitteren indított szavazása alapján a szavazók között kiemelkedő 87%-os aránnyal nyert a Git a verziókezelő rendszerek között.

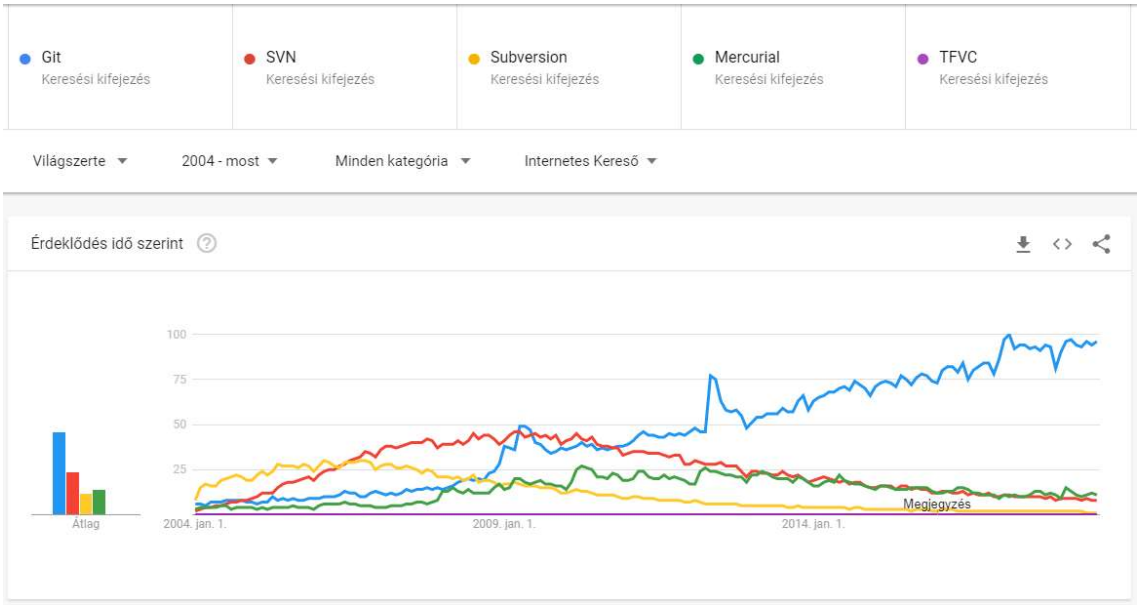
- A különböző verziókezelő rendszereket érintő internetes keresések száma (2016-os adatok) [27]



2-7. kép: GitPrime: „How Did SVN Manage to Lose?" Version control interest over time

A GitPrime 2016-os verziókezelő rendszerek iránti érdeklődés és internetes keresések számát vizsgáló kutatása alapján egyedül a Git volt képes hosszútávú és szinte töretlen felívelést felmutatni. Ráadásul ezt az eredményt több mint 10 éves időintervallum (Google Trends, GitHub, Atlassian és GitLab) adatainak a feldolgozásával kapták meg.

- Google Trends mindenkori statisztikája 5 kiemelt verziókezelővel [28]



2-8. kép: Google Trends adatok

A háttérismeretek figyelembevételével az aktuális helyzet megismerése céljából saját keresést is végeztem. Ehhez a Google Trends keresési statisztikáit néztem meg 2004-től 2018 harmadik negyedévéig bezárólag. A kereshető kulcskifejezések száma korlátozott, ezért öt kiemelt verziókezelő rendszer nevének keresési számait szemlélteti a fenti kép. Jól láthatóan ezek közül jelenleg is a leggyakrabban a Gitre keresnek rá a Google keresőjével.

Az előzőek alapján jól látható, hogy a verziókezelő rendszerek közül a Gitet a legcélszerűbb vizsgálni. Egyrészt mivel manapság is az egyik legnépszerűbb választás, másrészt mivel a népszerűségének köszönhetően az idők során sok nagyobb múltú projektnek lett Gites tükör repositoryja vagy váltott teljes egészében Git verziókezelésre.

2.5 Git alapfogalmak

Git esetén a verziókezelés alapját egy adatbázis szolgáltatja, amely nyilvántartja a különböző fájlverziókat és az azokhoz kapcsolódó egyéb információkat. Ez az adatbázis a *repository*. Az adatbázisban található információ legfrissebb, legújabb állapotát a *head* mutatja. A teljes kódbázis aktuális helyi verziója a *working copy*. Önálló, a többi funkciótól jól elkülöníthető, vagy időben hosszantartó fejlesztések kezelésére van lehetőség külön ágakat (*branch*) használni, melyeken párhuzamosan, de egymástól elszeparálva végezhetőek módosítások. Minden working copyn végzett módosítási egységet tudatni kell a verziókezelővel, hogy azokat eltárolja az adatbázisában. Egy ilyen

fájlmódosítási halmazt jelentő egység, kiegészülve egyéb metainformációkkal, a *commit*. A commit információi közé tartozik, hogy ki végezte a módosult és verziókövetett fájlokon a módosításokat (*author*), illetve ki hozta létre magát az adott commitot (*committer*). Az author és a committer személye általában megegyezik, azonban ez nem minden esetben van így, ezért szükséges a megkülönböztetésük.

Az angol nyelvről nem egyértelműen lefordítható, vagy magyarul szokatlanul hangzó szavak okozta kellemetlenségek elkerülése érdekében legtöbbször az angol megnevezés lesz a későbbiekben használva. Ezért az alábbi szószedetben táblázatosan is megtalálhatóak az előzőekben említett Githez kapcsolódó alapfogalmak. Fontos megjegyezni, hogy a szószedet nem tartalmaz minden alapfogalmat, csak a jelen munka szempontjából releváns és későbbiekben használt kifejezéseket.

Eredeti angol kifejezés	**A kontextuson belüli értelmezés magyar leírása**
repository	A verziókezelt fájlok összes verzióját tároló adatbázis.
working copy	A kódbázis helyi verziója, állapota.
head	A repositoryban lévő legfrissebb verzió hivatkozzása
history	A repository által eltárolt előzmény információk.
branch	A kódbázis egy virtuális másolata, amelyen akár huzamosabb ideig is lehet elkülönített változtatásokat végrehajtani anélkül, hogy azok közvetlenül bekerülnének egy másik branchbe.
commit	Egy verziókezelt egységbe zárt fájlmódosítások halmaza.
author	A commit tartalmát (fájl módosítások) előállító felhasználó azonosítója.
committer	A commitolást végző felhasználó azonosítója.

3 Téma felmérése, előzetes megfigyelések és az első eredmények kiértékelése

A kutatást segítő saját program megírása előtt a témakör előzetes felmérése érdekében megvizsgáltam pár projektet. Egy harmadik féltől származó program segítségével (GitStats), automatizáltan már feldolgozható formában kinyertem a Git repositoryjának előzményeit, majd a kapott adatokat manuálisan feldolgoztam Microsoft Excel segítségével. A feldolgozás eredményeként meghatároztam, hogy mit és hogyan fogok majd mérni a saját magam által implementált eszközzel.

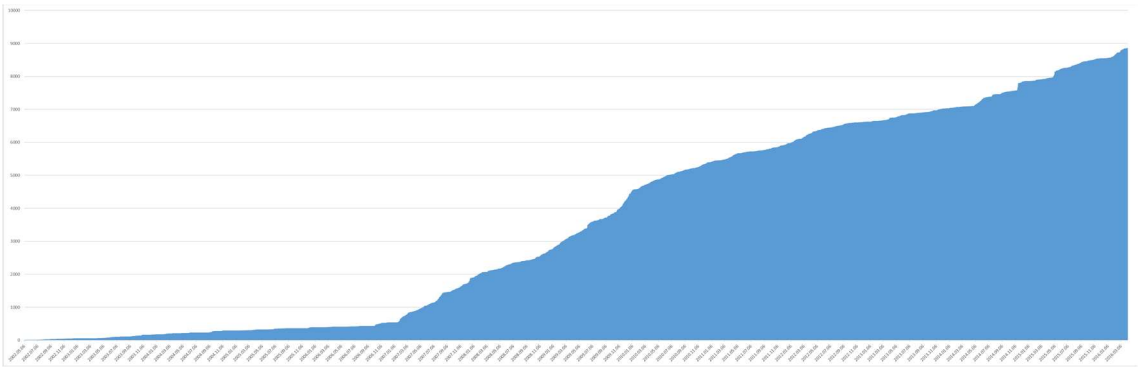
3.1 Mérési adatok validálása

Az előzetes vizsgálatot, valamint annak eredményeinek ellenőrzését, validálását az ipari partner számos repositoryján végeztem. Ezáltal könnyen tudtam visszajelzést szerezni a mért adatok nagyságrendi helyességével kapcsolatban. Megerősítést kaptam arról, hogy azokban az esetekben, ahol a mérések során kevés, vagy csak időszakos commit információk voltak, ott ténylegesen kevesen, vagy csak időszakosan fejlesztettek. Azokban az esetekben, ahol ritka, de kiugró mértékű a módosított sorok száma, és mindez alacsony commit szám mellett, ott valójában az aktív fejlesztés más ágon vagy más repositoryban történt és a vizsgált helyre csak időszakosan kerültek át az addig összegyűlt módosítások.

Előrevetítve a későbbi automatikus méréseket, fontos megjegyezni, hogy a manuális méréshez használt harmadik féltől származó eszköz (GitStats) implementációs döntései némiképp különbözhetnek a későbbiekben automatizált mérésekhez használt GitRepoAnalyzer program működésétől. Ebből kifolyólag a manuális és az automatikus vizsgálat adatait összevetve kis mértékben különböznek a pontos commit számok, és ebből adódóan némiképp a mérési eredmények is. Nagyságrendi, lényegi eltérés azonban nem látható.

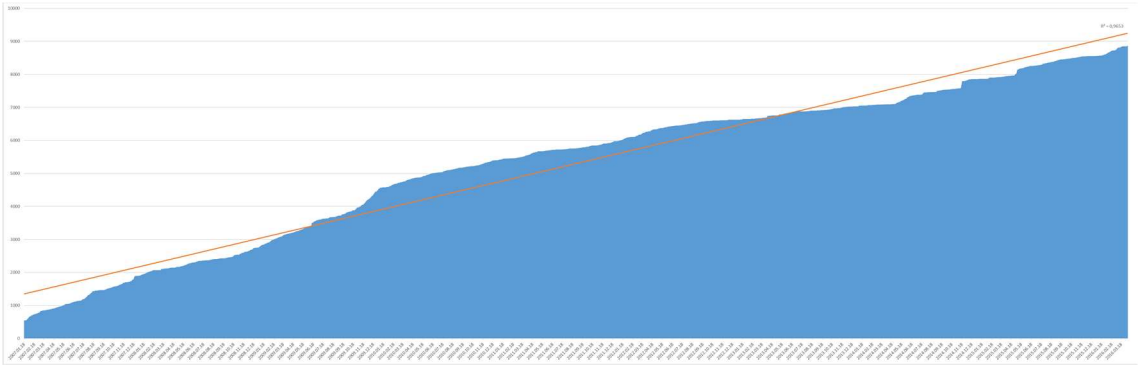
3.2 Termelékenységbecslés lineáris regresszióval

Miután az előzetes munka eredményeit validálták a projektben résztvevők, összefüggést keresve összevetettem a kapott eredmények diagramjait. Megvizsgáltam, hogy hogyan tudnám jól jellemezni a grafikonokat, és azt találtam, hogy a magasabb fokszámú polinomokkal való közelítések, a nagyobb szabadsági fokuk miatt jobban illeszkednek a görbére, mint a logaritmus, az exponenciális és a lineáris függvénnyel való közelítések.



3-1. kép: internal_titan_eclipse vagy más néven oldTitanEclipse
(kumulált commitok a teljes időintervallumon)

Ha a diagram idősoros x-tengelyének elején eltekintünk a projekt indulásának kezdeti értékeitől, akkor egy lényegesen jobb, méghozzá a lineáris regresszióra nézve 0,9653 R-négyzet értékű adatsort kapunk.



3-2. kép: internal_titan_eclipse vagy más néven oldTitanEclipse
(kumulált commitok az aktív fejlesztés időintervallumán)

A lineáris becslés ráadásul összhangban van az alábbi, már korábban is említett, Lehman törvényekkel:

- „*Conservation of Organisational Stability (invariant work rate)*”
- „*Conservation of Familiarity*”

Ezen törvények szerint a szoftver átlagos fejlődésének mértéke közel állandó a teljes életciklusa alatt.

Ezek alapján a lineáris közelítés megfelelően jó becslést jelent. Főleg azt figyelembe véve, hogy nagyon egyszerűen számolható, könnyen érthető és értelmezhető, skálázható, ezáltal jól használható.

3.3 Témához kapcsolódó projektek

Az alábbiakban említésre kerül néhány hasonlóan Gittel és azon belül history vizsgálattal foglalkozó szoftveres projekt. Ezek felhasználhatóságát megnézve arra jutottam, hogy részben a kimeneti formátum nehézkes feldolgozhatósága, részben a nem megfelelő számú mért metrikák és részben a projektek elhagyatottsága miatt a munkám során jobb nem ezekre támaszkodni a szükséges adatok méréséhez. Ennek ellenére mindenképp említést érdemelnek, már csak azért is, mert felépítésük, működésük és forráskódjuk hasznos tapasztalatot jelentett a későbbi munkához.

- GitStats: „Git history statistics generator”
  - Link:
    - http://gitstats.sourceforge.net/
    - https://github.com/hoxu/gitstats
- Git-Stats: „Local git statistics including GitHub-like contributions calendars”
  - Link:
    - https://github.com/IonicaBizau/git-stats
    - https://github.com/IonicaBizau/git-stats-html
- gitstat: „Gitstat is a web-based statistics and monitoring system for git”
  - Link:
    - https://sourceforge.net/projects/gitstat/
- Gitinspector: „The statistical analysis tool for git repositories”
  - Link:
    - https://github.com/ejwa/gitinspector

3.4 Miért kellett saját megoldás?

Az előbb említett projekteket átnézve egyik sem tűnt megfelelőnek ahhoz, hogy a jelenlegi téma vizsgálatához elegendő mennyiségű és minőségű adatot tudjon szolgáltatni. Csak listaszerűen néhány észrevétel, ami miatt nem tűntek jó választásnak a felhasználhatóság tekintetében:

- Az output lehetőség nem volt elég rugalmas vagy korszerű.
- Kevés szempontból, kevés metrikával dolgozott.
- Látszólag hosszú ideje nem fejlesztett projekt, segítségkérés vagy bug esetén megnehezítette volna a használatot és komoly kockázatot jelentett volna a jelen munka elkészültére.

- Szükség volt az anonim módú mérések támogatására az ipari partner méréseihez. Emellett a nyilvános repositoryk esetén a jobb visszakövethetőség és a futásidő javítása érdekében célszerű parametrizálni az anonimizáló opciót.
- Keretrendszert vagy telepítést igénylő előfeltételek szükségessége a projekt használatához.
- Nem feleltek meg a könnyen üzembehelyezhetőség elvárásának, hogy minél több helyen és minél kisebb befektetés mellett lehessen használni.

Ezen tapasztalatok és észrevételek alapján körvonalazódott, hogy kell egy új, ezekből a szempontokból a témának jobban megfelelő szoftver eszköz, ami képes egyedi szempontokból, metrikákkal és formában előállítani a későbbi elemzéshez szükséges adatokat.

3.5 Implementációs döntések

A fejlesztés során felmerültek döntést igénylő esetek. A méréseket végző programban ezek az alábbi módon lettek kezelve:

- **Egy repositoryban több branch is lehet.**
  - Van azonban minden repositoryban egy alapértelmezett branch, kezdetben egységesen master névvel. Általában ez a branch tekinthető a projekt fő ágának. Az esetek túlnyomó többségében fixen létezik és ennek az ágnak a tartalma jellemzi legjobban a projekt aktuális állapotát.
  - A branchek között a commitok úgymond átmásolhatóak (cherry-pick), tehát külön ellenőrzést igényelne, hogy egy commitot ne vegyen figyelembe többször is a program.
  - Branchek nem csak bármikor létrehozhatóak, de törölhetőek is. Ennek nyomon követése tehát további erőforrást igényelne.
  - **Ezek alapján, a program fixen és kizárólag a master branch historyának vizsgálatára épül.**
- **A Git a history lekérdezésekor közvetlenül megadja a hozzáadott sorok és a törölt sorok számát, a módosított sorokat viszont nem különbözteti meg.**
  - Ez a program szempontjából egy elfogadott kompromisszum.
  - Illetve ebből kifolyólag a program a sorok számát „Calculated” megkülönböztetéssel kezeli és a törölt és hozzáadott sorok összegét érti ez alatt.

    - **Tehát valójában egy sor módosítása a forráskódban, a program számára az eredeti sor törlésével és egy új sor hozzáadásával egyenértékű, ami ezáltal két sorműveletet jelent, tehát duplán számolódik.**
- **A Git az alapfogalmakat tartalmazó táblázat szerint tehát különbséget tesz az author és a committer között.**
    - Ez egyszerű és általános esetben nem is feltétlenül látszik, mivel ilyenkor a kettő egy és ugyan az a felhasználó, ezért sok esetben nem is különböztetik meg egymástól a kettőt.
    - **Előfordulhat azonban, hogy eltérőek lesznek. Ilyen esetben, mivel a program célja, hogy a termelékenységet, a befektetett munkát és a teljesítményt elemezze, ezért az author értékét, tehát a módosítás tartalmát ténylegesen előállító felhasználót veszi alapul.**
- **A különböző kódállapotok automatikus összefésülése külön commitot jelent (merge commit).**
    - **Mivel ez esetben automatizmusról van szó, tehát nem jelent valódi munkát, ezért a program direkt nem kéri le a historyból az ilyen commitokhoz tartozó információkat.**

4 GitRepoAnalyzer program bemutatása

A Git repositoryk előzetes elemzéséből, illetve a már meglévő Git repositorykat vizsgáló programok megismeréséből származó tapasztalatok és észrevételek alapján szükségessé vált egy új, az egyedi igényeknek megfelelő, a repositoryk elemzését és ezáltal a kutatási munkát segítő program létrehozása.

4.1 A program célja és feladata

A program célja, hogy a felhasználó által megadott Git repositoryból kigyűjtse az előzményeket (history) leíró adatokat, majd azokat a termelékenység elemzése szempontjából hasznos módon összegezve, összehasonlítható és tovább feldolgozható módon írja le a kimeneti fájlokban.

4.2 A program felépítése

A programot C# nyelven, ASP.NET Core keretrendszer felhasználásával írtam. Az ASP.NET Core egy platformfüggetlen, nagy teljesítményű, nyílt forráskódú keretrendszer, modern, web alapú alkalmazások, szolgáltatások, API-k (Application Programming Interface) létrehozásához. A program készítésekor a 2.1-es (az aktuálisan elérhető legújabb) verziót használtam, melynek hivatalos rövid ismertetője az alábbi linken érhető el: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.1. A jelen témához kapcsolódó kutatáshoz szükséges adatok előállításáról egy Web API gondoskodik. Ez képezi tehát a diplomamunka középpontját programozási szempontból.

Fontos szempont volt, hogy olyan széleskörben használt és szabványos formátumú kimeneti fájlokat állítsak elő a program kimeneteként, amelyeket később további feldolgozásra felhasználhatnak más egyéb, akár hasonló kutatási célra is. Az említett Web API képezi a program törzsét és látja el a fő funkciókat.

Annak érdekében, hogy szemléletesebbek és áttekinthetőbbek legyenek az eredmények készült egy szintén web alapú UI (User Interface) is az adatok megjelenítéséhez. Ez egyfajta plusz kényelmi funkcionalitást biztosít, melynek segítségével egyrészt dinamikusan megnézhető az eredményhalmaz egy bizonyos része, másrészt diagrammok segítségével könnyebben áttekinthetőek az eredmények. A webes megjelenítő használatakor azonban szem előtt kell tartani annak technológiai korlátait, tehát túl nagy adatmennyiség esetén drasztikusan megnőhet a megjelenítés futásideje vagy böngésző szintű korlátok miatt akár meg is hiúsulhat a megjelenítés. Ilyen esetben megoldást

jelenthet az egyszerre kisebb adatmennyiségek kezelése, vagy a korszerű és sokféleképpen feldolgozható JSON formátumú kimeneti fájl egyéb más feldolgozása.

4.3 A program futtatása és a futtatásához szükséges környezet

A program működéséhez az alábbi előfeltételek teljesülése szükséges:

- Feltelepített .NET Core 2.1 Runtime csomag a program futtatásához. (Itt letölthető: https://dotnet.microsoft.com/download/dotnet-core/2.1.)
- Repository elemzéshez szükség van feltelepített Git keretrendszerre. (Itt letölthető: https://git-scm.com/downloads.) Valamint ennek szerepelnie kell a környezeti változók között, hogy az adott platform parancssor vagy terminál alkalmazásából Git paranccsal elérhető legyen.
- A kimeneti eredményfájlok vizualizálásához, vagyis a GitRepoAnalyzer UI felületéhez szükség van egy web böngészőre. (A megjeleníthető adat mérete függ a böngésző típusától. Ajánlott Google Chrome böngészőt használni.)

A program Windows operációs rendszerű környezetben a GitRepoAnalyzer.exe futtatásával indítható el. A program az elindításától kezdve egy helyi webszerverként üzemel, és a képernyőn megjelenő URL-eken (melyeknek része a port is) keresztül érhető el.

4-1. kép: Elérhető HTTP és HTTPS protokollokon keresztül is

4.4 A program használata

Egy web böngésző segítségével az előzőleg említett URL-ek bármelyikének a meghívásakor az alábbi felület részletek lesznek láthatóak.

4.4.1 Menüpontok és funkciók bemutatása egy általános szcenárión keresztül

A következőkben a felületen megjelenő menüpontok kerülnek bemutatásra.

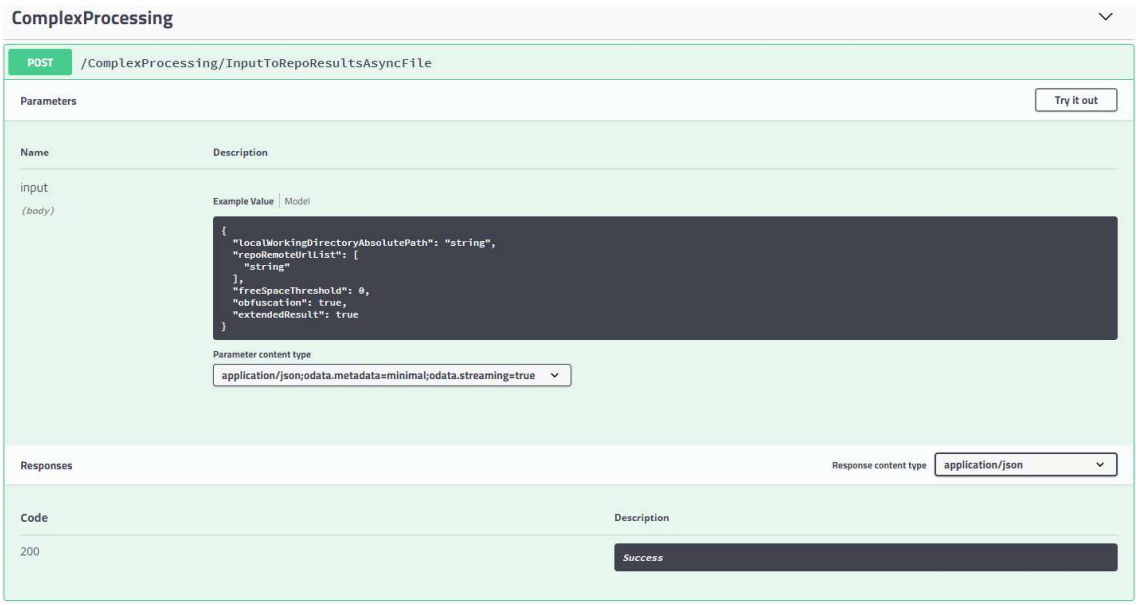
4-2. kép: Menüpontok

A „Home” menüpont a weboldal kezdő felületére navigál.

Az „API (Swagger Doc)” menüpont az API, illetve annak generált felületére navigál, ahol méréseket lehet indítani az ott megadott bemeneti adatoknak megfelelően, majd ennek elkészülte után az eredmény fájlokat letölteni.

Egy általános szcenárió e menüpont használata esetén az alábbiak szerint épül fel:

Ismert elérésű Git repositoryról szeretnénk egy lépésben megtudni repository szintű commitokkal és módosított sorokkal kapcsolatos információkat. Ez esetben a „ComplexProcessing” funkcióhalmazt lenyitva az „InputToRepoResultsAsyncFile” nevű funkciót kell használni.



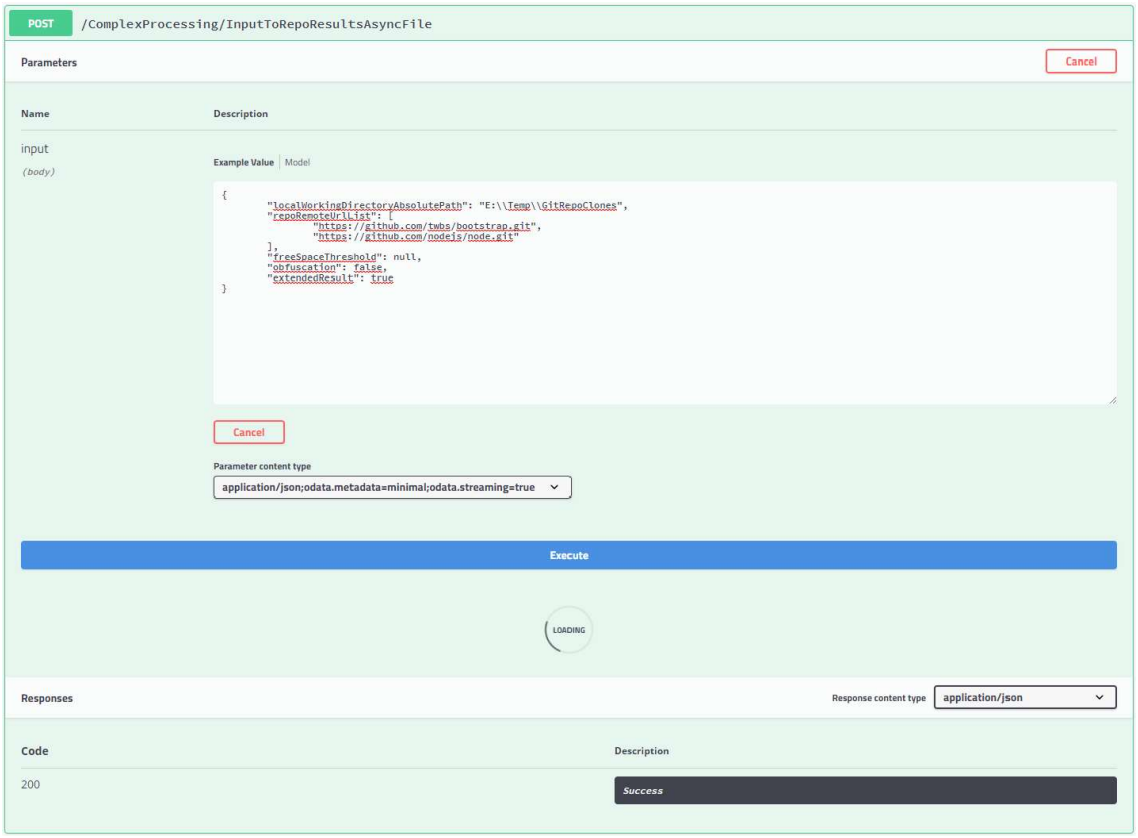
4-3. kép: A Git repositorykból kiindulva a cél a commitok és a sorok számának vizsgálata

A „Try it out” gombra kattintva szerkeszthetővé válik a bemeneti adatok beviteli mezője.

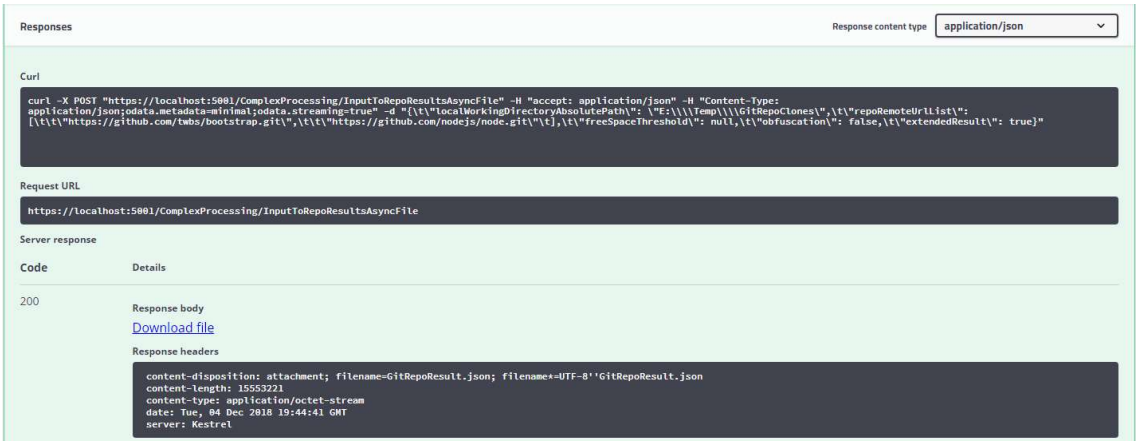
```
{
        "localWorkingDirectoryAbsolutePath": " C:\\ClonedGitRepos ",
        "repoRemoteUrlList": [
                "https://github.com/twbs/bootstrap.git",
                "https://github.com/nodejs/node.git"
        ],
        "freeSpaceThreshold": 0,
        "obfuscation": false,
        "extendedResult": true
}
```

4-4. kép: Példa beviteli adatok

Miután megfelelő a beviteli mező tartalma, az „Execute” gombbal elindul egy kérés a program felé. Ekkor megjelenik a felületen egy „Loading” felirat, ami jelzi, hogy a kérés megtörént, de válasz még nem érkezett rá. A kérést a program fogadja és elkezdi végrehajtani a meghívott műveletet, majd ha végzett a végrehajtással, akkor visszaküldi az eredményeket tartalmazó választ. A felületen keresztül egy JSON kiterjesztésű fájlként mindez letölthető lesz a „Download file” gombbal.



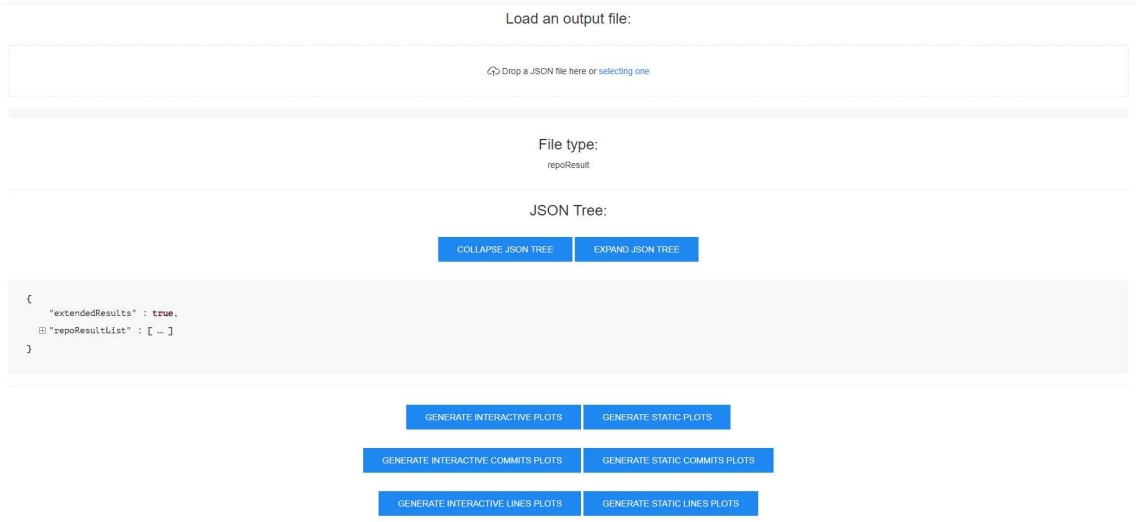
4-5. kép: A megfelelő beviteli adatok és az „Execute” gomb megnyomása után elindul a végrehajtás



4-6. kép: A válasz megérkezése után az eredmények fájlként letölthetőek a „Download file” lehetőséggel

Az „OutputViewer UI” menüponton belül a kimeneti fájlokban lévő adatok vizuálisan is megtekinthetőek. Első lépésként a fenti „Drop a JSON file here or selecting one” résznél vagy fogd és vidd (drag and drop) módszerrel, vagy tallózással betölteni a megjeleníteni kívánt eredmény fájlt.

Egyrészt lehetőség van a kimeneti JSON formátumú fájlt fa struktúrában áttekinteni. Az átláthatóság érdekében ekkor az adatszerkezeten belül az egyes csomópontok szabadon összecsukhatók és kinyithatók.



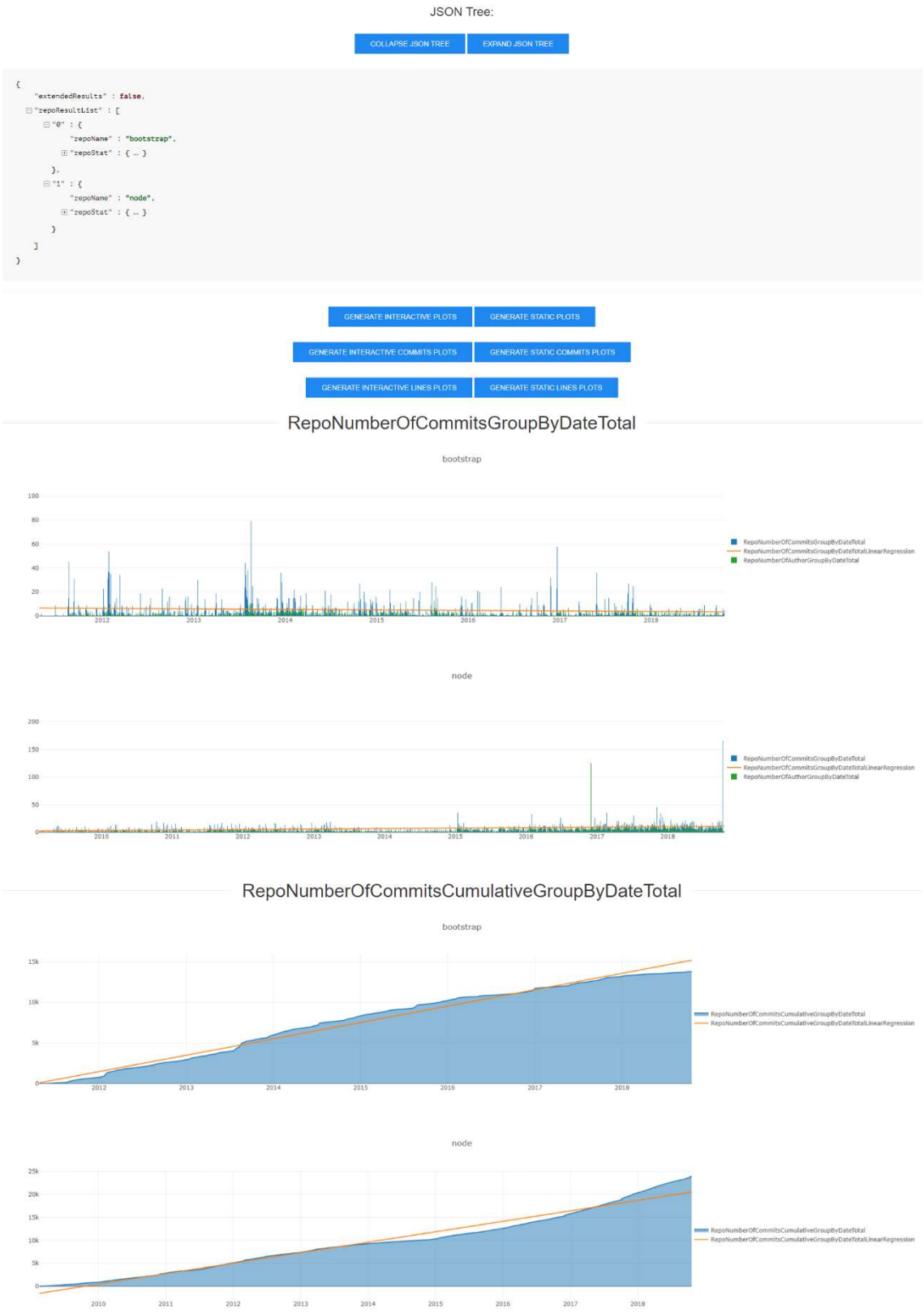
4-7. kép: Egy kimeneti JSON fájl betöltése után elérhetőek lesznek a további opciók, illetve a fa struktúra

Másrészt pedig lehetőség van diagrammokat generáltatni a böngészővel. Erre összesen hatféle lehetőség van.

- Generate Interactive Plots: Ez esetben commitok és sorok számáról is készül diagramm.
- Generate Interactive Commits Plots: Ez esetben csak a commitok számáról készül diagramm.
- Generate Interactive Lines Plots: Ez esetben csak a sorok számáról készül diagramm.

A felsorolt három eset mindegyike interaktív diagrammot (plot) jelent. Tehát lehetőség van interakcióba lépni a tartalmával. Vagyis például lehet nagyítani és kicsinyíteni a tartalmát, kijelölhető csupán a diagrammnak egy része, ami nagyobb részletességgel megnézhető.

A még nem említett másik három lehetőség az előző lehetőségek statikus verziói. Ez gyakorlatilag az elkészült interaktív diagrammoknak egy pillanatfelvétele, képe. Az így elkészült diagrammokkal nem lehet ugyan interakcióba lépni, de mivel képek, ezért egyszerűen kimásolhatóak vagy menthetőek. Az egyszerűbb kezelhetőség nem csak felhasználói szempontból igaz, hanem a böngésző is gyorsabban képes kezelni a statikus tartalmakat.



4-8. kép: Commitok számát ábrázoló statikus diagrammok

4.4.2 Kimeneti adatok

A kimeneti fájlok kiterjesztése JSON, tartalma pedig követi a szabványos JSON formátumot. Háromféle kimeneti fájlra van lehetőség a programmal.

1. GitHistory.json: A Git repositoryhoz tartozó history adatok kivonata. Ebben minden egyes commitra vonatkozóan benne vannak a program számára releváns információk:
   - addedRowsCount: a forráskód módosításával hozzáadott sorok száma
   - authorDateTime: a forráskód módosításának időpontja
   - authorName: a forráskód módosítását végző személy
   - calculatedRowsCount: az addedRowsCount és a removedRowsCount összege
   - fileChangedCount: a forráskód módosítása által változott fájlok száma
   - hash: a forráskód módosítás (commit) rövidített azonosítója.
   - removedRowsCount: a forráskód módosításával törölt sorok száma


```
{
	"repoHistoryList": [
		{
			"repoName": "linux",
			"commitInfoList": [
				{
					"hash": "1da177e4c3f4",
					"authorName": "Linus Torvalds",
					"authorDateTime": "2005-04-17T00:20:36+02:00",
					"addedRowsCount": 6718755,
					"removedRowsCount": 1,
					"fileChangedCount": 17291,
					"calculatedRowsCount": 6718756
				},
				{
					"hash": "a2755a80f40e",
					"authorName": "Linus Torvalds",
					"authorDateTime": "2005-04-21T01:24:21+02:00",
					"addedRowsCount": 1,
					"removedRowsCount": 1,
					"fileChangedCount": 1,
					"calculatedRowsCount": 2
				}
			]
		}
	],
	"extendedResult": false
}
```


4-9. kép: GitHistory.json példa fájl

2. GitRepoResults.json: Repositorykra jellemző adatokat tartalmaz. Ebben minden repositoryra vonatkozóan benne vannak a program számára releváns információk:
    - repoAverageNumberOfCommits: a teljes időintervallumra értelmezett átlagos commit szám
    - repoAverageNumberOfCommitsWithCommitDays: a commitos napokra értelmezett átlagos commit szám
    - repoAverageNumberOfLinesWithCommitDays: a commitos napokra értelmezett átlagos módosított sorok száma
    - repoAverageNumberOfLinesWithCommits: az egy commitban módosított sorok átlagos száma
    - repoAverageNumberOfLinesWithTotalDays: a teljes időintervallumra értelmezett átlagos módosított sorok száma
    - repoCommitsGroupByDate: a commitok dátumok szerint csoportosítva
    - repoData: a repositoryt jellemző adatok
    - repoDateListFromFirstCommitToLastCommit: az első és utolsó commit dátuma között lévő összes dátum listája
    - repoDateOfFirstCommit: a legelső commit időpontja
    - repoDateOfLastCommit: a legutolsó commit időpontja
    - repoDateOfMaxNumberOfAuthors: melyik nap volt a legtöbb módosítást végző felhasználó
    - repoDateOfMaxNumberOfCommits: melyik nap volt a legtöbb commit
    - repoDateOfMaxNumberOfLines: melyik nap volt a legtöbb módosított sor
    - repoMaxNumberOfDaysBetweenCommits: az egymást követő commitos napok között eltelt napok számának maximuma
    - repoName: a repository neve
    - repoNumberOfAuthorGroupByDate: minden commitos dátumra megadja, hogy aznap hány felhasználó módosított
    - repoNumberOfAuthorGroupByDateTotal: az első és utolsó commit közti összes napra megadja, hogy aznap hány felhasználó módosított
    - repoNumberOfCommitDays: azon napok száma, amelyeken volt commit

- repoNumberOfCommitsCumulativeGroupByDateTotal: minden commitos dátumra megadja, hogy kumulatívan hány commit volt az adott dátumig (az aznapot is beleértve)
- repoNumberOfCommitsCumulativeGroupByDateTotalLinearRegression: lineáris regresszió a teljes időintervallumon értelmezett kumulatív napi commit darabszámok alapján
  - intercept: az egyenes Y-tengellyel vett metszéspontja
  - slope: az egyenes meredeksége
  - rSquaredValue: lineáris regresszió R-négyzet értéke
- repoNumberOfCommitsGroupByDate: minden commitos dátumra megadja, hogy aznap hány commit volt
- repoNumberOfCommitsGroupByDateLinearRegression: lineáris regresszió a commitos dátumokra értelmezett napi commit darabszámok alapján
  - intercept: az egyenes Y-tengellyel vett metszéspontja
  - slope: az egyenes meredeksége
  - rSquaredValue: lineáris regresszió R-négyzet értéke
- repoNumberOfCommitsGroupByDateTotal: az első és utolsó commit közti összes napra megadja, hogy aznap hány commit volt
- repoNumberOfCommitsGroupByDateTotalLinearRegression: lineáris regresszió a teljes időintervallumon értelmezett napi commit darabszámok alapján
  - intercept: az egyenes Y-tengellyel vett metszéspontja
  - slope: az egyenes meredeksége
  - rSquaredValue: lineáris regresszió R-négyzet értéke
- repoNumberOfDaysBetweenCommits: az egymást követő commitos napok között eltelt napok száma
- repoNumberOfDaysBetweenFirstCommitAndLastCommit: az első és utolsó commit közt eltelt napok száma
- repoNumberOfLinesCumulativeGroupByDateTotal: az első és utolsó commit közti összes napra megadja, hogy kumulatívan hány commit volt az adott dátumig (az aznapot is beleértve)

- repoNumberOfLinesCumulativeGroupByDateTotalLinearRegression: lineáris regresszió a teljes időintervallumon értelmezett kumulatív napi módosított sorok száma alapján
  - intercept: az egyenes Y-tengellyel vett metszéspontja
  - slope: az egyenes meredeksége
  - rSquaredValue: lineáris regresszió R-négyzet értéke
- repoNumberOfLinesGroupByDate: minden commitos dátumra megadja, hogy aznap hány sor módosult
- repoNumberOfLinesGroupByDateTotal: az első és utolsó commit közti összes napra megadja, hogy aznap hány sor módosult
- repoTotalNumberOfCommits: az összes commit darabszáma
- repoTotalNumberOfLines: az összes módosult sor darabszáma

3. GitAuthorResults.json: Authorokra jellemző adatokat tartalmaz. Ebben minden authorra vonatkozóan benne vannak a program számára releváns információk:
   - authorAverageNumberOfCommitsWithCommitDays: az authorhoz tartozó commitok átlagos napi száma a commitos napok arányában
   - authorAverageNumberOfCommitsWithTotalDays: az authorhoz tartozó commitok átlagos napi száma az első és utolsó commitja közti teljes időintervallum arányában
   - authorAverageNumberOfLinesWithCommitDays: az authorhoz tartozó módosított sorok átlagos napi száma a commitos napok arányában
   - authorAverageNumberOfLinesWithCommits: az authorhoz tartozó módosított sorok átlagos száma a commitok arányában
   - authorAverageNumberOfLinesWithTotalDays: az authorhoz tartozó módosított sorok átlagos napi száma az első és utolsó commitja közti teljes időintervallum arányában
   - authorCommits: az authorhoz tartozó commitok
   - authorCommitsGroupByDate: az authorhoz tartozó commitok dátumok szerint csoportosítva
   - authorDateListFromFirstCommitToLastCommit: az authorhoz tartozó első és utolsó commit közti összes dátum
   - authorDateOfFirstCommit: az authorhoz tartozó első commit időpontja

- authorDateOfLastCommit: az authorhoz tartozó utolsó commit időpontja
- authorDateOfMaxNumberOfCommits: az authorhoz tartozó commitos napok közül annak a dátuma, amelyik napon a legtöbb hozzá tartozó commit volt
- authorDateOfMaxNumberOfLines: az authorhoz tartozó commitos napok közül annak a dátuma, amelyik napon a legtöbb hozzá tartozó módosított sor volt
- authorMaxNumberOfDaysBetweenCommits: az authorhoz tartozó commitos napok között eltelt napok száma közül a maximum
- authorName: az author neve
- authorNumberOfCommitDays: az authorhoz tartozó commitos napok száma
- authorNumberOfCommits: az authorhoz tartozó commitok száma
- authorNumberOfCommitsGroupByDate: az authorhoz tartozó commitok dátum szerint csoportosítva (csak a commitos napokat figyelembevéve)
- authorNumberOfCommitsGroupByDateTotal: az authorhoz tartozó commitok dátum szerint csoportosítva (az első és utolsó commitos nap közötti összes napot figyelembevéve)
- authorNumberOfDaysBetweenCommits: az authorhoz tartozó commitos napok között eltelt napok száma
- authorNumberOfDaysBetweenFirstCommitAndLastCommit: az authorhoz tartozó első és utolsó commit közti napok száma
- authorNumberOfLines: az authorhoz tartozó módosított sorok száma
- authorNumberOfLinesGroupByDate: az authorhoz tartozó módosított sorok száma dátum szerint csoportosítva
- authorNumberOfLinesGroupByDateTotal: az authorhoz tartozó módosított sorok száma dátum szerint csoportosítva (az első és utolsó commitos nap közötti összes napot figyelembevéve)
- authorStats: az authort jellemző adatok
- repoData: a repositoryt jellemző adatok
- repoName: a repository neve

```
{
	"repoResultList": [{
		"repoName": "linux",
		"repoStat": {
			"repoDateOfFirstCommit": "1970-01-01T00:00:00+01:00",
			"repoDateOfLastCommit": "2037-04-25T00:00:00+02:00",
			"repoCommitsGroupByDate": {
				"1970-01-01T00:00:00+01:00": [{
					"hash": "224426f168aa",
					"authorName": "Ursula Braun",
					"authorDateTime": "1970-01-01T01:00:01+01:00",
					"addedRowsCount": 2,
					"removedRowsCount": 1,
					"fileChangedCount": 1,
					"calculatedRowsCount": 3
				}],
				"2037-04-25T00:00:00+02:00": [{
					"hash": "12ca45fea91c",
					"authorName": "Daniel Vetter",
					"authorDateTime": "2037-04-25T10:08:26+02:00",
					"addedRowsCount": 19,
					"removedRowsCount": 23,
					"fileChangedCount": 1,
					"calculatedRowsCount": 42
				}]
			},
			"repoDateListFromFirstCommitToLastCommit":
				["1970-01-01T00:00:00+01:00",
				"2037-04-25T00:00:00+02:00"],
			"repoNumberOfDaysBetweenFirstCommitAndLastCommit": 24586,
			"repoNumberOfCommitDays": 5030,
			"repoNumberOfDaysBetweenCommits": [11582,
				2446],
			"repoMaxNumberOfDaysBetweenCommits": 11582,
			"repoNumberOfCommitsGroupByDate": {
				"1970-01-01T00:00:00+01:00": 1,
				"2037-04-25T00:00:00+02:00": 1
			},
			"repoNumberOfCommitsGroupByDateTotal": {
				"1969-12-31T00:00:00+01:00": 0,
				"1970-01-01T00:00:00+01:00": 1,
				"2037-04-25T00:00:00+02:00": 1
			},
			"repoNumberOfAuthorGroupByDate": {
				"1970-01-01T00:00:00+01:00": 1,
				"2037-04-25T00:00:00+02:00": 1
			},
			"repoNumberOfAuthorGroupByDateTotal": {
				"1969-12-31T00:00:00+01:00": 0,
				"1970-01-01T00:00:00+01:00": 1,
				"2037-04-25T00:00:00+02:00": 1
			},
			"repoDateOfMaxNumberOfCommits": "2008-01-30T00:00:00+01:00",
			"repoDateOfMaxNumberOfAuthors": "2007-05-08T00:00:00+02:00",
```


4-10. kép: GitRepoResults.json példa fájl (1. rész)

```
                        "repoTotalNumberOfCommits": 725665,
                        "repoTotalNumberOfLines": 80840648,
                        "repoAverageNumberOfCommits": 29.515374603432846,
                        "repoAverageNumberOfCommitsWithCommitDays":
                                144.26739562624255,
                        "repoNumberOfLinesGroupByDate": {
                                "1970-01-01T00:00:00+01:00": 3,
                                "2037-04-25T00:00:00+02:00": 42
                        },
                        "repoNumberOfLinesGroupByDateTotal": {
                                "1969-12-31T00:00:00+01:00": 0,
                                "1970-01-01T00:00:00+01:00": 3,
                                "2037-04-25T00:00:00+02:00": 42
                        },
                        "repoDateOfMaxNumberOfLines": "2005-04-17T00:00:00+02:00",
                        "repoAverageNumberOfLinesWithTotalDays": 3288.0764662816237,
                        "repoAverageNumberOfLinesWithCommitDays": 16071.699403578528,
                        "repoAverageNumberOfLinesWithCommits": 111.40215939862057,
                        "repoNumberOfCommitsGroupByDateTotalLinearRegression": {
                                "slope": 0.0020177864809578657,
                                "intercept": 4.7073157047062821,
                                "rSquaredValue": 0.038822460691981031
                        },
                        "repoNumberOfCommitsCumulativeGroupByDateTotal": {
                                "1969-12-31T00:00:00+01:00": 0,
                                "1970-01-01T00:00:00+01:00": 1,
                                "2037-04-25T00:00:00+02:00": 725665
                        },
                        "repoNumberOfLinesCumulativeGroupByDateTotal": {
                                "1969-12-31T00:00:00+01:00": 0,
                                "1970-01-01T00:00:00+01:00": 3,
                                "2037-04-25T00:00:00+02:00": 80840648
                        },
                        "repoNumberOfCommitsCumulativeGroupByDateTotalLinearRegression":
                        {
                                "slope": 40.297076780285764,
                                "intercept": -234202.57809666981,
                                "rSquaredValue": 0.77687267978542973
                        },
                        "repoNumberOfLinesCumulativeGroupByDateTotalLinearRegression": {
                                "slope": 4574.6623498734589,
                                "intercept": -25569017.43845731,
                                "rSquaredValue": 0.80227072668949928
                        },
                        "repoNumberOfLinesGroupByDateTotalLinearRegression": {
                                "slope": 0.19357317545666605,
                                "intercept": 908.11718004991144,
                                "rSquaredValue": 0.00095307778870842082
                        }
                }
        }],
        "extendedResults": true
}
```


4-11. kép: 2. GitRepoResults.json példa fájl (2. rész)

5 Repositoryk elemzése

5.1 A vizsgált repositoryk általános jellemzői

A manuális vizsgálat és annak validálása után, az ipari partner repositoryjain túl további 50 különböző nyílt forráskódú, GitHubon elérhető projekten végeztem méréseket. Ezen projektek kiválasztása részben véletlenszerű, részben direkt választott volt. Véletlenszerűek abban a tekintetben, hogy jórészük közismert és a saját területükön belül széles körben használt keretrendszerek, eszközök, alkalmazások. Másrészt direkt választottak abban a tekintetben, hogy az előbbi válogatás eredményeként létrejött projekthalmazt mind programozási nyelv, mind a projekt fejlesztési körülményei, mind a repository méret alapján történő célzott kereséssel bővítettem ki, hogy minél változatosabb vagy szemléletesebb összehasonlítások születhessenek.

Programozási nyelv tekintetében a mért repositoryk között nagyrészt az a jellemző, hogy van egy fő programozási nyelv, ami kiemelkedő túlsúlyban van a repository tartalmát tekintve, de van pár kevésbé egyértelműen egy nyelven íródó projekt is. A GitHub a repository tartalma alapján képes az adott projekt forráskódjában használt programozási nyelveket, valamint azok teljes kódbázishoz viszonyított százalékos arányát kimutatni. Ez az arány minden később szereplő GitHub repository esetén külön fel lesz tüntetve. A kiemelkedő arányú nyelvek között megtalálható a C, C++, C#, Haskell, Java, JavaScript, PHP, Python, Ruby, Scala, TypeScript, ugyanakkor voltak repositoryk, ahol kevésbé egyértelmű lenne kiemelni egy meghatározó programozási nyelvet.

A projektek irányítottságát, hátterét tekintve van folyamatosan és szervezetten fejlődő ipari hátterű projekt, időszakosan és ellenőrzötten fejlődő egyetemi projekt, és van független, szervezeti háttér nélkül fejlődő projekt. A projektek ezen szempont szerinti felosztása nem feltétlenül egyértelmű, de ez is jelzi a mért projekthalmaz változatosságát.

A repository mérete általában jó előrejelzést jelent az elemzés időigényére nézve. Eszerint a következő méthatárokat állítottam fel, amely alapján csoportosíthattam az elemzéseket várható időigény szerint. Fontos megjegyezni, hogy a repository mérete a mérések idejében értendő, hiszen később nyilván változhat.

- Kisméretűnek számít egy repository, ha nagyságrendileg 100 MB vagy annál is kisebb méretű.

- Közepes méretűnek számít egy repository, ha nagyságrendileg a 100 MB és 1 GB méret közé esik.
- Nagyméretűnek számít egy repository, ha nagyságrendileg 1 GB méretet meghaladó.

Viszonyításképpen a repository méretét elosztva az első és utolsó commit közt eltelt napok számával átlagosan kis és közepes méretű repositoryk esetén 0,11 MB/nap méret, míg nagyméretű repositoryk esetén átlagosan 0,29 MB/nap eredmény jött ki a mérések alapján. A valós érték természetesen sok tényezőtől függ, és ebből adódóan ettől lényegesen eltérhet. Reális körülmények között lehetséges, hogy több tízévnyi fejlesztés után is 1 GB alatt marad a repository mérete (lásd cpython), de az is lehetséges, hogy néhány éven belül eléri ezt a méretet (lásd TypeScript).

A nagyméretű repositoryk elemzésekor operációsrendszer szintű limitációkba is lehet ütközni. Ilyen tapasztalat volt például, hogy a Windows operációs rendszer egy bizonyos határérték felett nem képes szöveget (jelen esetben például a GitHistory.json fájlban tárolt tartalmát) másolni vagy vágólapra helyezni. Megjegyzésképpen ez empirikus tapasztalat alapján az általam használt futtatókörnyezetben ez a határérték valahol 1,5 millió sor környékén volt, amit a nagyméretű repositoryk kimeneti fájljai könnyedén túlléphetnek.

Viszonyításképpen az előzményeket tartalmazó GitHistory.json fájl tartalma nagyságrendileg a következőképpen becsülhető. A fájl szerkezeti kerete vehető 10 sornak, valamint minden egyes commit további 10 sort jelent. Tehát becsülve az 1 millió sor 100 000 commitot jelent. Napi 10 committal számolva, az év összes napját beleértve is több, mint 27 év lenne. Napi 30 committal számolva, ami nem irreális eset egy aktívabban fejlesztett projektnél, pedig már csak valamivel több, mint 9 év.

Megjegyzésképp a nagyméretű repositoryk nagyméretű kimeneti fájljait nem minden általános szövegszerkesztésre alkalmas program képes megnyitni és használható módon kezelni abban az esetben, ha szükség lenne azok manuális megnyitására. Tapasztalati példa erre a Notepad++ nevű program, mely helyett ilyen esetben egy jobb alternatíva például a Sublime Text, ami jobban kezeli a nagyméretű szöveges (jelen esetben JSON) fájlokat.

5.2 A repository mérések bemutatása

Az alábbi mérési eredmények célja az egyes projektek fejlesztésének, fejlődésének vizsgálata.

A manuálisan elvégzett mérések alapján a termelékenység lineárisan becsülhető. Ennek alátámasztásához nagyobb projekthalmazon is meg kell vizsgálni a lineáris illeszkedést, illetve annak „jóságát” az R-négyzet értékkel. A lineáris illeszkedés egyenletes termelést, egyenletes termelékenységet jelent. A vizsgálati eredmények elérése érdekében minden repositoryn az alábbi automatizált műveletsort hajtjuk végre a GitRepoAnalyzer program segítségével:

- Kinyerjük a repository historyját.
- A history adatokat csoportosítjuk dátum szerint, mellyel megkapjuk, hogy egy adott napon hány darab commit volt.
- A dátumokhoz tartozó darabszámokat az idő előrehaladtával kumuláltan összegezzük. Ezzel megkapjuk, hogy egy adott dátumig bezárólag egészen addig hány darab commit volt.
- Az eddig megkapott adatok függvényesíthetők, tehát az így kapott függvényhez kereshetünk legjobban illeszkedő egyenest.
- A kapott legjobban illeszkedő egyenesnek vizsgálhatjuk a meredekségét, de ami jelen esetben a sejtés szempontjából fontosabb, az az egyenesre számított lineáris regresszió R-négyzet értéke, mellyel bizonyos jósági fokot kapunk az adott repository valós fejlődése és az általánosított lineáris sejtés viszonyáról. Az R-négyzet értéke a [0-1] intervallumon belül minél közelebb áll az 1-hez, annál jobban illeszkedik a valódi fejlesztési trend a lineárishoz.

A megvizsgált repositoryk száma és mérete alapján, valamint figyelembevéve a nyomtatott megjelenítés korlátait, szükség volt a repositoryk, ezáltal az eredmények kategorizálására és válogatására. A szövegtörzs, igazodva a lapmérethez, valamint a formai megkötésekhez, nem tartalmaz minden mért eredményt. Azonban fontos megjegyezni, hogy az eredmények efféle kiválogatásánál nem a kutatási konklúzió torzítása vagy kozmetikázása volt a cél. Épp ezért nem az R-négyzet értékek alapján lettek kiválogatva a törzsanyagban szereplő repositoryk és azok eredményei. A döntést minden esetben a leghosszabb commit nélküli időszak hossza, valamint a commitok összesített száma határozta meg. Ezáltal olyan mérési eredmények kerültek ki a fejezetből, amelyeknél a többihez viszonyítva a leghosszabb időintervallumon nem volt commit, tehát ebből a szempontból látszólag az adott időszakban megállt a projekt, valamint azok a repositoryk, amelyek esetében a többihez képest szignifikánsan kevesebb számú

commit volt. Utóbbi esetben a kisebb adathalmazon végzett lineáris becslés és megfigyelés az ésszerű terjedelmi határok miatt tehát kisebb jelentőségűnek lett tekintve.

A kutatás során elvégzett további mérési eredmények, amelyek az aktuális fejezetből kimaradtak, megtalálhatóak a függelékben.

Mivel az egyes projektekről belső információk hiányában nehéz megmondani, hogy mekkora részben tekinthetők iparilag, céges szinten, vagy fizetett fejlesztők által fejlesztett projekteknek, ezért az ilyen irányú kategorizálás nem lehetne kellően precíz. Emiatt a kategorizálás alapjául az egyes repositoryk egyértelműen meghatározható információi vagy a bizonyíthatóan döntő mértékű tulajdonságai szolgáltak. A repositorykhoz kapcsolódó projektek információi alapján az alábbi kategóriák figyelhetők meg:

- Ipari partnerhez kapcsolódó repositoryk
- GitHub projektekhez kapcsolódó repositoryk
- OsmoCom projekthez kapcsolódó repositoryk

A repositoryk bizonyíthatóan döntő mértékű tulajdonságai alapján a GitHub projektek esetén tovább kategorizálhatóak fő programozásnyelv alapján. Ennek alátámasztására minden ilyen repository esetén százalékos arányban látható a repositoryban található forráskódok jelenlegi programnyelvi aránya. Ez alapján külön kategóriát alkotnak az alábbi fő programozási nyelvű repositoryk:

- C
- C++
- C#
- Haskell
- Java
- JavaScript
- PHP
- Python
- Ruby
- Scala
- TypeScript

A programnyelvi kategorizálás lehetőséget ad arra, hogy megnézzük, hogy a jelenlegi adathalmazon látható-e valamiféle eltérés a nyelvből adódóan, vagy nyelvtől függetlenül hasonló trendeket és eredményeket mutatnak.

Először az összehasonlíthatóság végett táblázatos formában egymás mellett lesznek láthatóak a repositoryk különböző jellemző adatai, majd utána részletesebben, egyesével is látható lesz a commitok és sorok idősoros alakulása diagramszerűen, valamint az adatokhoz legjobban illeszkedő egyenes.

5.3 Eredmények

Az összes mérési eredmény megtalálható a függelékként a következő elérhetőségen:

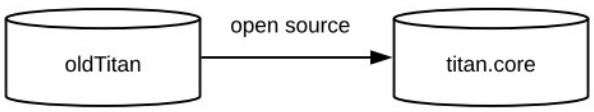
http://compalg.inf.elte.hu/~attila/TestingAtScale.htm [29]

5.3.1 Ipari partnerhez kapcsolódó repositoryk

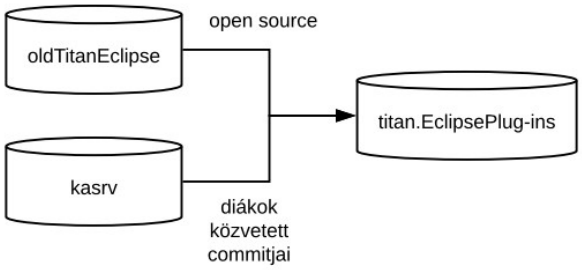
Az ipari partner által ipari repositoryk eredményeit is megvizsgáltam. A saját fejlesztésű GitRepoAnalyzer program segítségével anonimizált mérési eredményeket kaphattam néhány belső elérésű, valamint időközben nyílt forráskódúvá vált céges projekt repositoryjából. A mért repositoryk nevein és eredményein kívül további háttérinformációkat kaptam az egyes projektek életciklusa során bekövetkezett eseményekről. Egyrészt érdekes volt kideríteni, hogy az eredményekben megfigyelt változások mögött volt-e valóban valamilyen projektet érintő változás, és ha igen, akkor milyen jellegű. Másrészt érdekes volt a háttérinformációkból kiindulva megnézni, hogy az egyes események jelentettek-e kimutatható változást a repository eredményeiben.

A kapott információ szerint, a repositoryk tartalmát tekintve a kasrv, az oldTitanEclipse és a titan.EclipsePlug-ins jellemzően Java fájlokat, az oldTitan és a titan.core jellemzően C és C++ fájlokat, míg a Merged, a System2 és az SFTF repository leginkább TTCN-3 fájlokat tartalmaz.

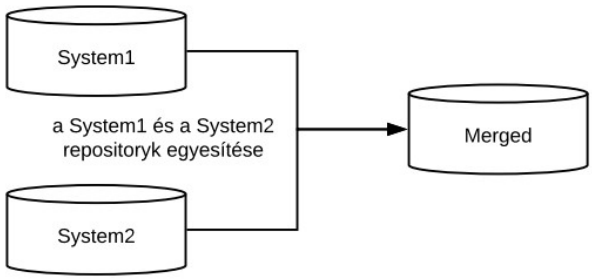
A vizsgált repositoryk az alábbiak szerint állnak kapcsolatban egymással:



A projekteknél a nyílt forráskódúvá válás egyben új repositoryt is jelentett.



A kasrv repositoryn keresztül az egyetemi diákok is közvetetten fejleszthetik a titan.EclipsePlug-ins projektet. Fontos azonban a diákok commitjainak ellenőrzése, valamint a két repository közti szinkronizáció.



A System1 és System2 projektek kezdetben külön-külön fejlődtek. Majd 2012 decemberében egyesítve lettek.

A következőkben a kapott háttérinformációkat tüntettem fel. Tekintettel arra, hogy a felsorolás pontjai között lehetnek olyanok, amelyek egyszerre több, vagy akár mindegyik repositoryra is hatással lehettek, ezért a háttérinformációk nem az egyes diagramok mellé, hanem az alábbi helyre lettek összegyűjtve (természetesen azért némi struktúráltságra és kategorizálásra szükség volt az áttekinthetőség végett):

System2 és Merged:

- 2010.: Az addig házon kívül fejlesztett System-2 projektet átvette egy belső fejlesztőcsapat, akik úgy döntöttek, hogy újraírják a projektet.
- 2012. eleje: A System-2 új verziókezelő repositoryba került.
- 2012. december: System-1 és System-2 projekt egybe lett olvasztva a Merged nevű repositoryban.
- 2013. július: Egy "Titanium Quest" nevű esemény keretében többek között 10%-kal csökkentették a "fixme" és "todo" jellegű kommentek, 57%-kal a kör importálások és 50%-kal a nem használt importálások számát.
- 2014. első félév: A Merged projekt összes rendszer architektjét egyetlen vezető rendszer architektre váltották.
- 2014. július: Megrendezték a "Green Day" eseményt, melyen többek között a még megmaradt, de nem használt importok nagy részét törölték.

- 2014. december: Az előzőekhez hasonlóan egy "Black Thursday" eseményen a refaktorálás eredményeképpen a teljes kódbázis 0,6%-át törölték.
- 2016 április: A Gites repoitorykat átszervezték. Körülbelül 900 fájl és 640 ezer sor lett áthelyezve. A kódbázis valójában nem nőtt ezáltal.
- 2016 június: Verziókezelés alá került több millió sornyi generált help fájl, melyek a kódban lévő kommentek alapján generálódtak.
- 2017 január: Az előzőleg repositoryba rakott, generált help fájlok nem bizonyultak jó megoldásnak. Online elérhető tették a help fájlokat, majd törlésre kerültek a verziókezelőből.

oldTitan:

- A sorok száma esetén látható hirtelen ugrások mind hozzáadott tesztekhez köthetőek.
- 2009: Új funkció az XML kódolás támogatása. Ehhez kapcsolódóan bekerültek nagy mennyiségben szabványhoz kapcsolódó tesztek, valamint nagy adatmennyiségű konverziós teszt validált eredményei.
- 2009. március: Bekerült egy 2,5 millió soros XML fájl az XML dekódoló teljesítmény és terhelési teszteléséhez.
- 2010 augusztus: Nagyobb változások a tesztek között. Például több, mint egy millió sor törlésre került.
- 2011 körül: A tesztrendszer egy részének átalakítása egy új módszerűre. Az új módszer egyszerre, egy nagy forráskódhalmaz egyszeri lefordításával dolgozott és az adott forráskódrészletek után kommentként belegenerálta a teszt eredményét. Az egyszerre fordítással sok esetben elfedték egymást a hibák, azért a kipróbálás után nem jelentett jó megoldást.

titan.core:

- 2018. január: Bekerült a repositoryba egy nagyméretű teljesítményteszt csomag.

oldTitanEclipse:

- 2014. szeptember: A projekt új, opensource iránya miatt a Java csomagok nevei com.ericsson kezdetűről org.eclipse kezdetűre változtak. Az átnevezés miatt szinte minden fájl változott, miközben lényegében tartalmilag egyik sem.

titan.EclipsePlug-ins:

- 2015 év eleje: A projekt opensource repositoryba történő átmozgatása során több, mint 800 000 sort érintő commitok is történtek.
- 2017 nyara: Érkezett három diákmunkás a nyárra. Közülük egy azóta is a projekten dolgozik, egy másik csak azon a nyáron dolgozott, a harmadik pedig 2017 és 2018 nyarán is dolgozott a projekten diákként.
- 2017 december: Sikeresen legenerálhatóvá vált a TitanLoggerAPI, ami egyből be is került a verziókezelőbe. Ez körülbelül 77 ezer sort jelentett.
- 2018 június: TitanLoggerAPI frissítés.
- 2018 augusztus vége: AsciiDocra váltás miatt új dokumentációs fájlok kerültek be a repositoryba.
- 2018 október vége: TitanLoggerAPI frissítés.

Általánosabb érvényű háttérinformációk:

- Eredetileg az oldTitan és az oldTitanEclipse CVS-ben volt tárolva, ugyanabban a repositoryban. Később, ahogy az Eclipse része egyre inkább önálló termékké nőtte ki magát, szétváltak repository szinten is. A szétválasztáskor a verziókezelőben tárolt adatok mindkét esetben migrálva lettek Gitbe.
- 2017 vége: Új épületbe költözött az ipari partnercég. Az új épületre az addiginél jobban jellemzőek a nyitott irodák és nagy terek.

Általános adatok táblázata:

Repository neve	Első és utolsó commit között eltelt napok száma	Commitos napok száma	Leghosszabb commit nélküli időszak hossza napokban	Commitok száma összesen	Sorok száma összesen
kasrv	2 314	827	73	3 548	3 718 691
Merged	2 129	1 520	14	12 759	57 404 479
oldTitan	5 075	3 160	24	20 813	16 456 821
oldTitanEclipse	5 085	1 920	64	8 531	4 858 443
SFTF	1 157	237	59	317	318 266
System2	994	602	24	3 195	13 888 356
titan.core	1 502	442	129	990	3 106 109
titan.EclipsePlug-ins	1 379	709	200	4 370	1 293 102

Átlag értékek táblázata:

Repository neve	Átlag commit szám az összes napra vetítve	Átlag commit szám a commitos napokra vetítve	Átlag sorok száma az összes napra vetítve	Átlag sorok száma a commitos napokra vetítve	Átlag sorok száma egy commitra vetítve
kasrv	1,533	4,290	1 607,040	4 496,603	1 048,109
Merged	5,993	8,394	26 963,118	37 766,105	4 499,136
oldTitan	4,101	6,586	3 242,723	5 207,855	790,699
oldTitanEclipse	1,678	4,443	955,446	2 530,439	569,505
SFTF	0,274	1,338	275,079	1 342,895	1 003,994
System2	3,214	5,307	13 972,189	23 070,359	4 346,903
titan.core	0,659	2,240	2 067,982	7 027,396	3 137,484
titan.EclipsePlug-ins	3,169	6,164	937,710	1 823,839	295,904

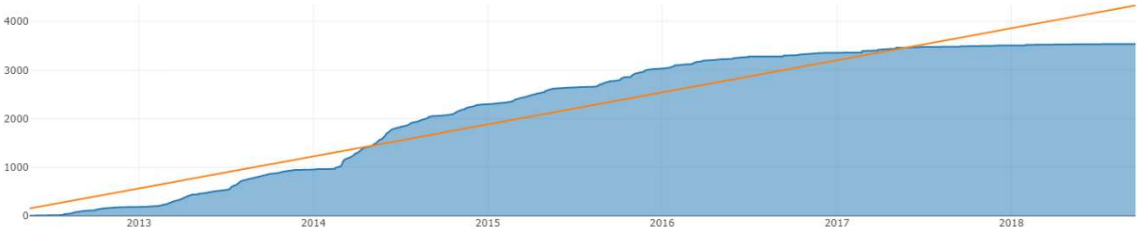
Kumulált commitok számához számított lineáris regressziós értékek:

Repository neve	Kumulált commit számok esetén vett lineáris regresszió meredeksége	Kumulált commit számok esetén vett lineáris regresszió metszéspontja	Kumulált commit számok esetén vett lineáris regresszió R-négyzet értéke
kasrv	1,805821	150,934887	0,915753
Merged	6,062922	444,731621	0,993736
oldTitan	4,624929	90,515018	0,980866
oldTitanEclipse	1,981024	-1 481,998340	0,952535
SFTF	0,220009	26,361193	0,964282
System2	3,317780	-437,379782	0,968115
titan.core	0,780098	-210,892262	0,928103
titan.EclipsePlug-ins	3,279566	-1 024,899126	0,852987

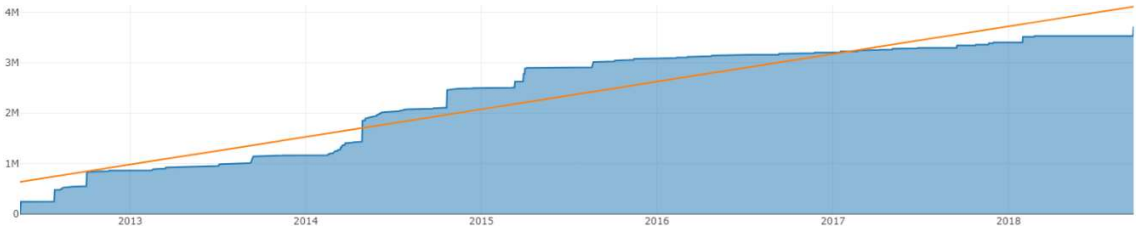
Kumulált sorok számához számított lineáris regressziós értékek:

Repository neve	Kumulált sorok száma esetén vett lineáris regresszió meredeksége	Kumulált sorok száma esetén vett lineáris regresszió metszéspontja	Kumulált sorok száma esetén vett lineáris regresszió R-négyzet értéke
kasrv	1 500,717412	637 998,205847	0,897161
Merged	17 712,564254	23 731 734,386794	0,821206
oldTitan	3 956,296361	-1 152 267,661551	0,933683
oldTitanEclipse	889,512046	175 351,923230	0,967906
SFTF	125,410449	183 705,840389	0,514850
System2	14 615,994350	-681 491,358991	0,921879
titan.core	1 564,972499	970 167,483538	0,755692
titan.EclipsePlug-ins	788,108542	55 229,348012	0,915318

5.3.1.1 kasrv

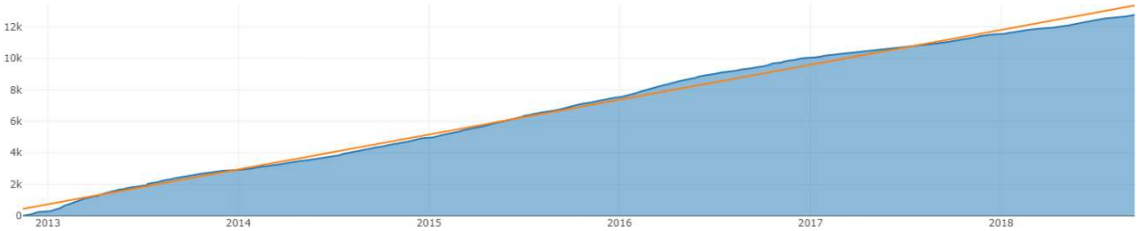


5-1. kép: kasrv (kumulált commitok)

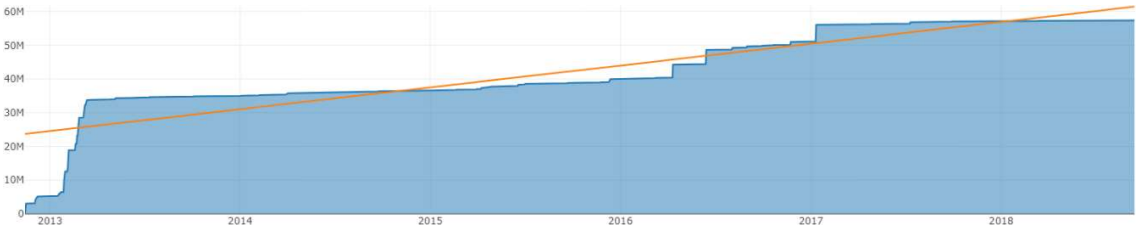


5-2. kép: kasrv (kumulált sorok)

5.3.1.2 Merged

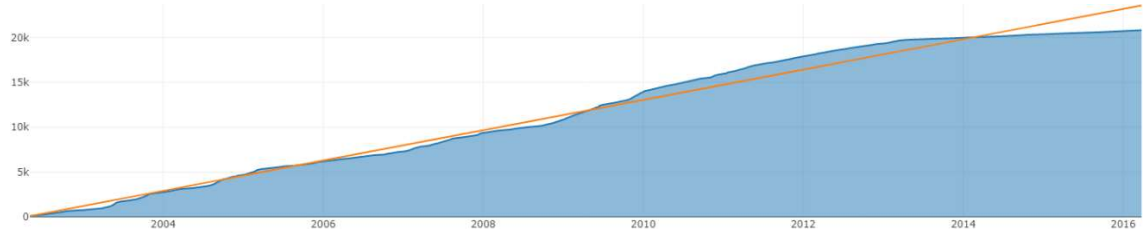


5-3. kép: Merged (kumulált commitok)

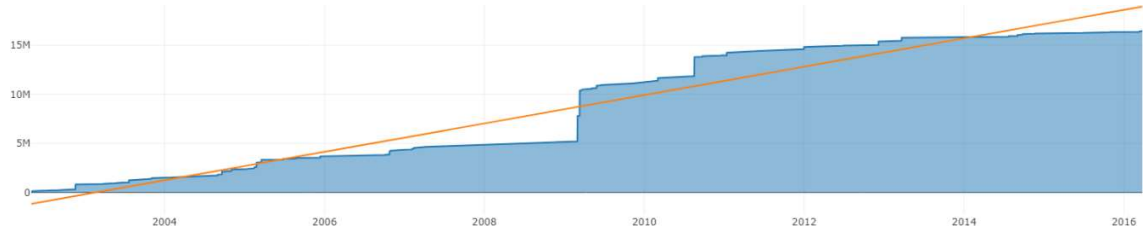


5-4. kép: Merged (kumulált sorok)

5.3.1.3 oldTitan

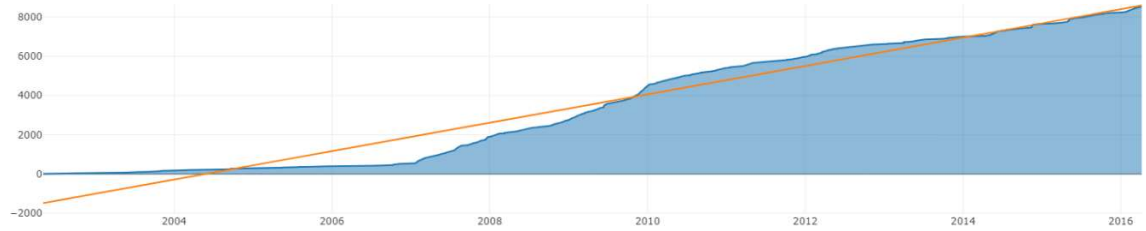


5-5. kép: oldTitan (kumulált commitok)

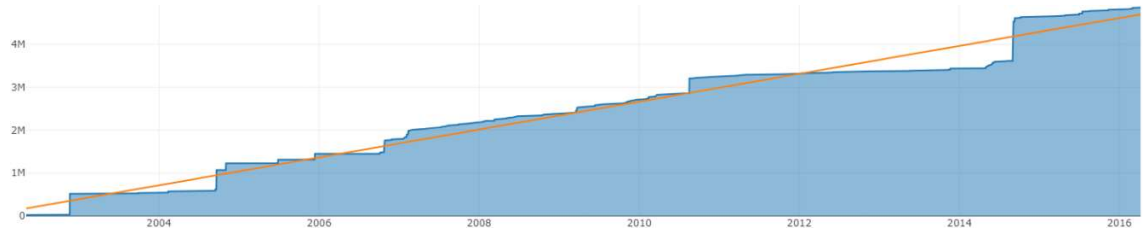


5-6. kép: oldTitan (kumulált sorok)

5.3.1.4 oldTitanEclipse



5-7. kép: oldTitanEclipse (kumulált commitok)



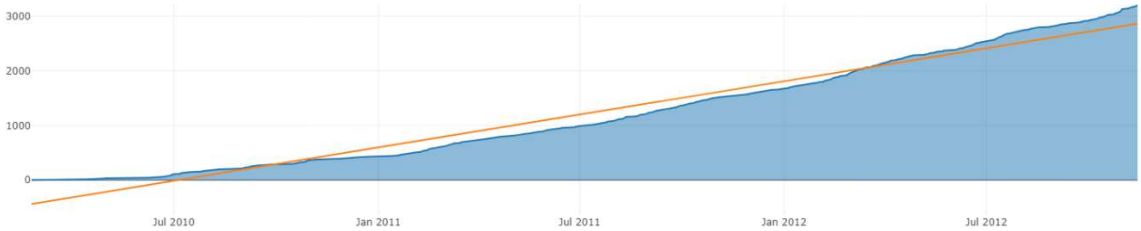
5-8. kép: oldTitanEclipse (kumulált sorok)

5.3.1.5 SFTF

5-9. kép: SFTF (kumulált commitok)

5-10. kép: SFTF (kumulált sorok)

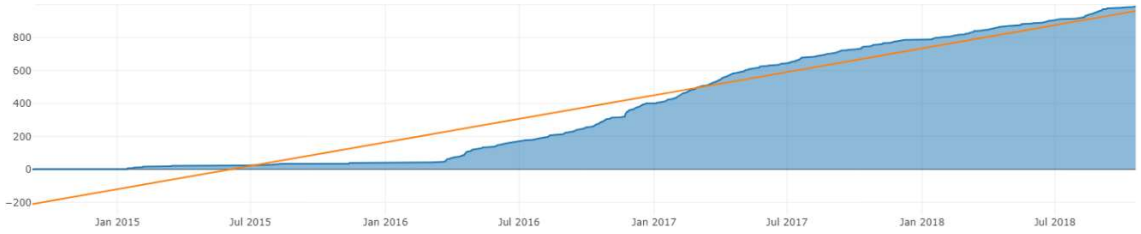
5.3.1.6 System2



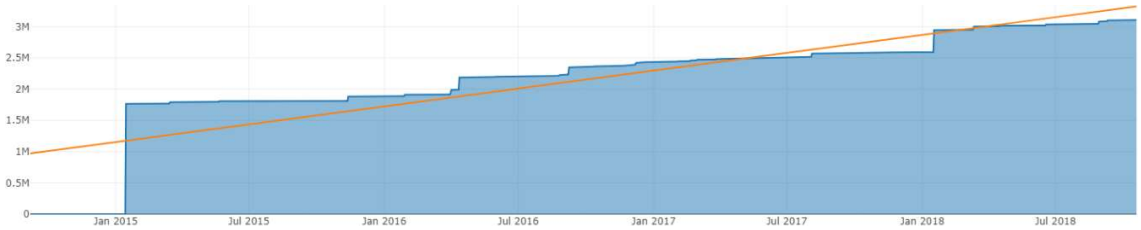
5-11. kép: System2 (kumulált commitok)

5-12. kép: System2 (kumulált sorok)

5.3.1.7 titan.core

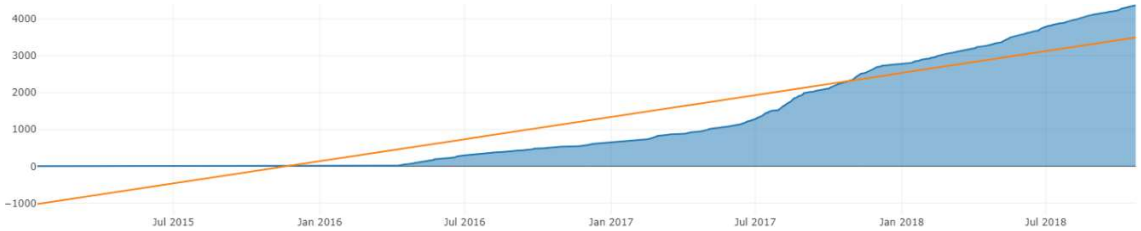


5-13. kép: titan.core (kumulált commitok)

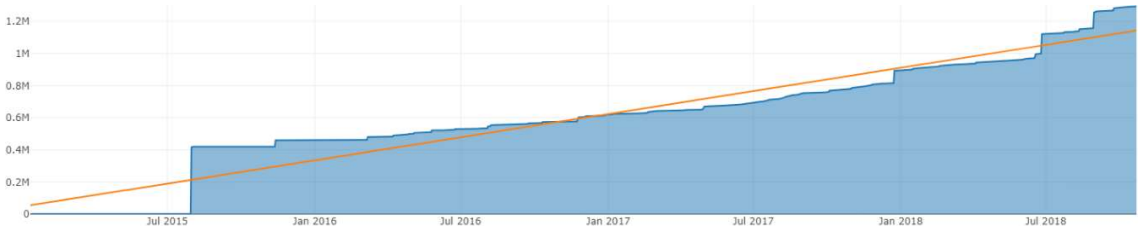


5-14. kép: titan.core (kumulált sorok)

5.3.1.8 titan.EclipsePlug-ins



5-15. kép: titan.EclipsePlug-ins (kumulált commitok)



5-16. kép: titan.EclipsePlug-ins (kumulált sorok)

5.3.1.9 Ipari partnerhez kapcsolódó repositoryk konklúziója

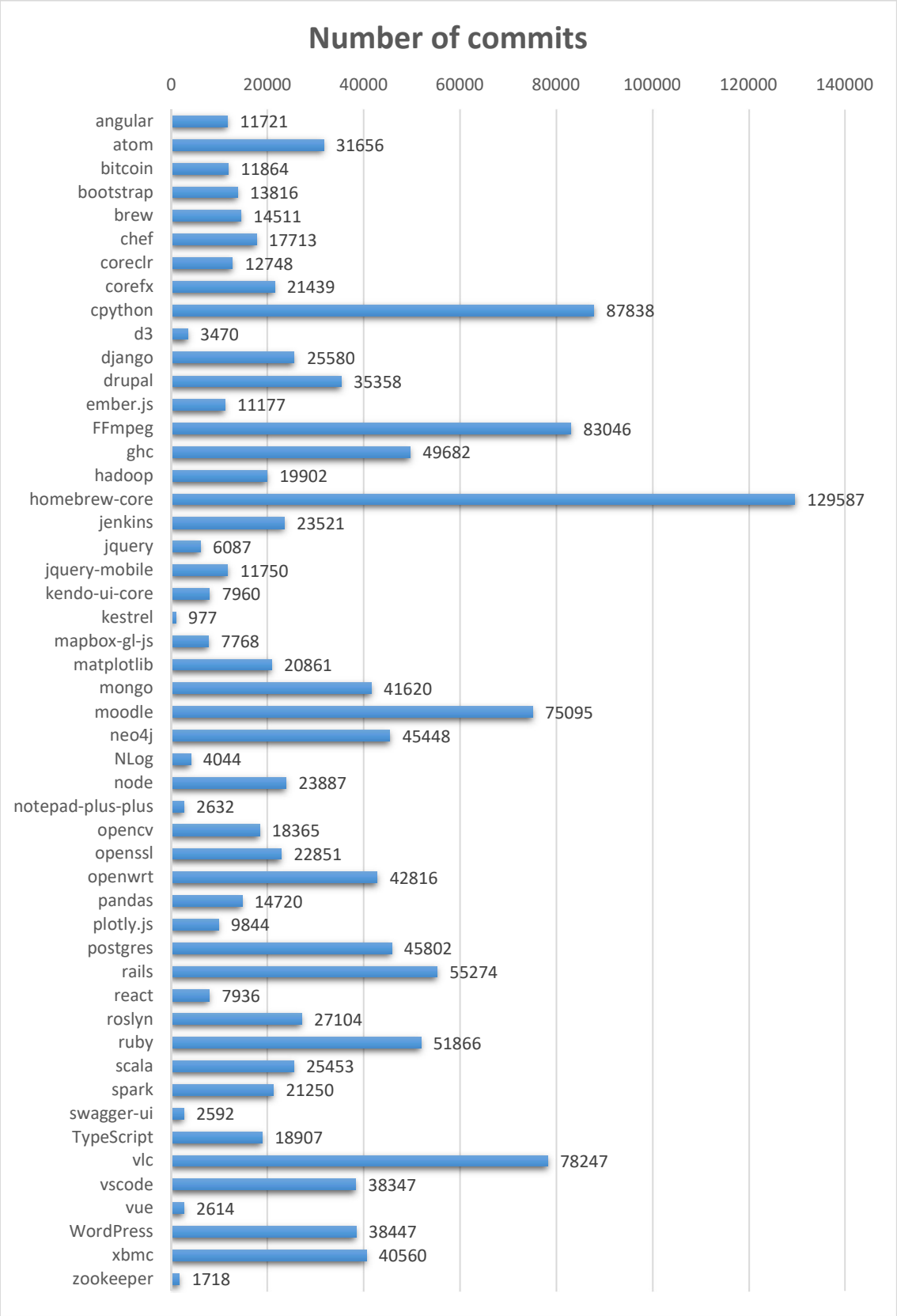
A projektek nyílt forráskódúvá válásának folyamatáról az alábbi megállapítások tehetőek:

- A régi és az új repositoryk között (a hosszabb-rövidebb tanulási, kísérletezési és átmeneti időszaktól eltekintve) jól látható a váltás.
- A régi és az új repositoryk közti váltáskor az illeszkedő egyenes meredeksége jelentősen megváltozott.
  - Az oldTitan – titan.core esetén jelentősen csökkent a meredekség.
  - Az oldTitanEclipse – titan.EclipsePlug-ins esetén jelentősen nőtt a meredekség.
- A nyílt forráskód jelentette sokkal nagyobb potenciális fejlesztői bázis nem hozott ugrásszerű javulást a mért adatokban.

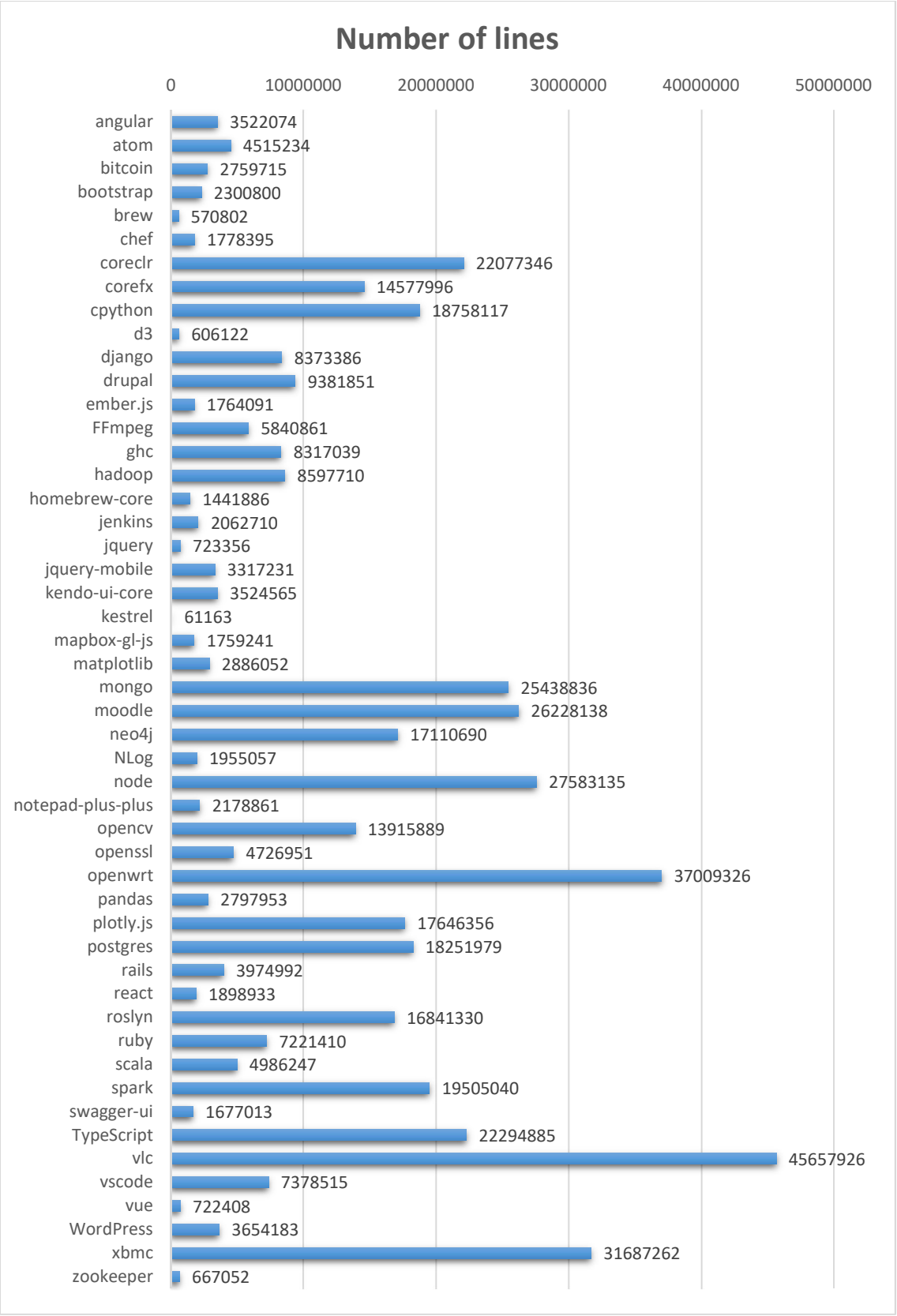
Az eredmények láttán az alábbi általános megállapítások tehetőek:

- A szervezett kódrefaktorálási napok és események a teljes repository adataihoz képest sem a commitok számában, sem a módosított sorok számában nem okoznak szignifikáns változást.
- A különböző programnyelvek szempontjából nem jelenik meg szignifikást különbség a repositoryk diagrammjain. Leginkább csak meredekségbeli különbségek láthatóak a különböző C, C++, Java és TTCN-3 alapú projekteknél.
- A vezetői átszervezés (architekt váltás) nem eredményezett látható változást.
- A cég új helyre költözése egyrészt jól sikerült, mert nem okozott komolyabb visszaesést a projektekben, másrészt az új, nyitott irodai környezet nem hozott jelentős különbséget a mért metrikákban.
- Az egyetemi hallgatók számára létrehozott kasrv repository időnként szinkronizálva lett a mindeközben folyamatosan (állandó munkaviszonnyal rendelkezők által) fejlesztett titan.EclipsePlug-ins repositoryval. Tehát emiatt nem csak a diákok munkája jelenik meg a repository tartalmában. A szinkronizációkból adódóan a titan.EclipsePlug-ins esetén az állandó előrehaladást időszakosan feljavította a hallgatók munkája. A másik oldalról viszont, pont amiatt, hogy állandó munka folyt a titan.EclipsePlug-ins repositoryban, és az ebből adódó változásokat időnként átszinkronizálták a kasrv repositoryba, ezért ott kevésbé látszódik a kapott diagrammon a diákok munkájára jellemző időszakosság (oktatási szünetek, zárthelyi dolgozatok, vizsgaidőszakok).

5.3.2 Kis és közepes méretű GitHub repositoryk



5-17. kép: 3.1.1 GitHub projektekhez kapcsolódó repositoryk esetén a commit számok



5-18. kép: 3.1.1 GitHub projektekhez kapcsolódó repositoryk esetén a sorok száma

Általános adatok táblázata:

Repository neve	Első és utolsó commit között eltelt napok száma	Commitos napok száma	Leghosszabb commit nélküli időszak hossza napokban	Commitok száma összesen	Sorok száma összesen
angular	1 493	1 363	7	11 721	3 522 074
atom	2 619	2 277	24	31 656	4 515 234
bootstrap	2 733	2 007	48	13 816	2 300 800
brew	3 439	2 763	14	14 511	570 802
chef	3 879	2 908	20	17 713	1 778 395
coreclr	1 359	1 296	3	12 748	22 077 346
corefx	1 444	1 416	3	21 439	14 577 996
cpython	10 299	8 164	40	87 838	18 758 117
django	4 847	4 127	12	25 580	8 373 386
drupal	6 729	5 223	49	35 358	9 381 851
ember.js	2 730	2 281	12	11 177	1 764 091
ghc	8 320	6 504	53	49 682	8 317 039
hadoop	3 441	2 873	13	19 902	8 597 710
homebrew-core	3 425	3 371	12	129 587	1 441 886
jenkins	4 365	3 684	8	23 521	2 062 710
kendo-ui-core	1 771	1 260	10	7 960	3 524 565
mapbox-gl-js	1 940	1 364	29	7 768	1 759 241
mongo	4 017	3 470	26	41 620	25 438 836
moodle	6 175	5 685	32	75 095	26 228 138
neo4j	4 165	3 020	34	45 448	17 110 690
node	3 533	3 221	19	23 887	27 583 135
notepad-plus-plus	4 087	1 642	37	2 632	2 178 861
opencv	3 084	2 736	9	18 365	13 915 889
openssl	7 242	4 751	14	22 851	4 726 951
openwrt	5 318	4 757	25	42 816	37 009 326
pandas	3 362	2 617	45	14 720	2 797 953
plotly.js	2 312	1 528	19	9 844	17 646 356
postgres	8 138	7 459	12	45 802	18 251 979
rails	5 078	4 821	12	55 274	3 974 992
react	1 971	1 670	9	7 936	1 898 933
roslyn	1 677	1 616	4	27 104	16 841 330
scala	5 727	4 603	17	25 453	4 986 247
spark	3 126	2 460	31	21 250	19 505 040
swagger-ui	2 644	849	50	2 592	1 677 013
TypeScript	1 565	1 473	4	18 907	22 294 885
vlc	7 013	6 377	49	78 247	45 657 926
vscode	1 072	1 055	2	38 347	7 378 515
vue	906	491	40	2 614	722 408
WordPress	5 666	4 722	32	38 447	3 654 183
xbmc	3 313	3 288	4	40 560	31 687 262
zookeeper	4 003	979	39	1 718	667 052

Átlag értékek táblázata:

Repository neve	Átlag commit szám az összes napra vetítve	Átlag commit szám a commitos napokra vetítve	Átlag sorok száma az összes napra vetítve	Átlag sorok száma a commitos napokra vetítve	Átlag sorok száma egy commitra vetítve
angular	7,85	8,60	2 359,06	2 584,06	300,49
atom	12,09	13,90	1 724,03	1 982,97	142,63
bootstrap	5,06	6,88	841,86	1 146,39	166,53
brew	4,22	5,25	165,98	206,59	39,34
chef	4,57	6,09	458,47	611,55	100,40
coreclr	9,38	9,84	16 245,29	17 034,99	1 731,83
corefx	14,85	15,14	10 095,57	10 295,19	679,98
cpython	8,53	10,76	1 821,35	2 297,66	213,55
django	5,28	6,20	1 727,54	2 028,93	327,34
drupal	5,25	6,77	1 394,24	1 796,26	265,34
ember.js	4,09	4,90	646,19	773,38	157,83
ghc	5,97	7,64	999,64	1 278,76	167,41
hadoop	5,78	6,93	2 498,61	2 992,59	432,00
homebrew-core	37,84	38,44	420,99	427,73	11,13
jenkins	5,39	6,38	472,56	559,91	87,70
kendo-ui-core	4,49	6,32	1 990,16	2 797,27	442,78
mapbox-gl-js	4,00	5,70	906,83	1 289,77	226,47
mongo	10,36	11,99	6 332,79	7 331,08	611,22
moodle	12,16	13,21	4 247,47	4 613,57	349,27
neo4j	10,91	15,05	4 108,21	5 665,79	376,49
node	6,76	7,42	7 807,28	8 563,53	1 154,73
notepad-plus-plus	0,64	1,60	533,12	1 326,96	827,83
opencv	5,95	6,71	4 512,29	5 086,22	757,74
openssl	3,16	4,81	652,71	994,94	206,86
openwrt	8,05	9,00	6 959,26	7 779,97	864,38
pandas	4,38	5,62	832,23	1 069,15	190,08
plotly.js	4,26	6,44	7 632,51	11 548,66	1 792,60
postgres	5,63	6,14	2 242,81	2 446,97	398,50
rails	10,88	11,47	782,79	824,52	71,91
react	4,03	4,75	963,44	1 137,09	239,28
roslyn	16,16	16,77	10 042,53	10 421,62	621,36
scala	4,44	5,53	870,66	1 083,26	195,90
spark	6,80	8,64	6 239,62	7 928,88	917,88
swagger-ui	0,98	3,05	634,27	1 975,28	647,00
TypeScript	12,08	12,84	14 245,93	15 135,70	1 179,19
vlc	11,16	12,27	6 510,47	7 159,78	583,51
vscode	35,77	36,35	6 882,94	6 993,85	192,41
vue	2,89	5,32	797,36	1 471,30	276,36
WordPress	6,79	8,14	644,93	773,86	95,04
xbmc	12,24	12,34	9 564,52	9 637,25	781,24
zookeeper	0,43	1,75	166,64	681,36	388,27

Kumulált commitok számához számított lineáris regressziós értékek:

Repository neve	Kumulált commit számok esetén vett lineáris regresszió meredeksége	Kumulált commit számok esetén vett lineáris regresszió metszéspontja	Kumulált commit számok esetén vett lineáris regresszió R-négyzet értéke
angular	8,129325	-389,245762	0,998159
atom	13,894895	-227,029575	0,966364
bootstrap	5,529524	82,147899	0,982913
brew	4,446559	-1 786,539155	0,964509
chef	4,931720	-1 731,429174	0,989441
coreclr	10,242405	-1 118,178678	0,987032
corefx	16,849183	-2 013,008487	0,987048
cpython	9,770079	-14 912,882121	0,963244
django	5,563023	-792,830504	0,992707
drupal	5,712684	-5 201,985774	0,948961
ember.js	4,312132	-148,667336	0,996084
ghc	6,539264	-4 996,694469	0,990795
hadoop	6,333203	-3 127,088431	0,971700
homebrew-core	34,535089	-21 939,061938	0,877979
jenkins	5,523732	1 205,531706	0,991382
kendo-ui-core	5,393632	-143,410710	0,956491
mapbox-gl-js	4,174024	243,439446	0,984488
mongo	11,145602	-3 320,933602	0,996543
moodle	13,393050	-2 029,189755	0,991969
neo4j	11,446684	-5 058,385784	0,972103
node	6,235704	-1 521,232672	0,971055
notepad-plus-plus	0,606941	-57,253513	0,958808
opencv	6,327812	540,565164	0,971776
openssl	2,433738	575,020865	0,910099
openwrt	8,613126	-4 010,931747	0,992839
pandas	4,872183	-1 549,626794	0,980834
plotly.js	4,639738	-1 833,412646	0,944022
postgres	5,664687	496,881721	0,993322
rails	12,424616	-5 488,500480	0,984398
react	4,021715	197,396737	0,997999
roslyn	18,039433	-2 778,818478	0,989622
scala	4,551146	302,494349	0,991952
spark	7,971638	-4 380,196201	0,938173
swagger-ui	1,018399	-476,374131	0,893892
TypeScript	11,939220	551,128840	0,997739
vlc	12,393602	-10 990,299339	0,979620
vscode	35,833310	-1 588,996277	0,995969
vue	2,531917	723,784497	0,922915
WordPress	7,630480	-4 849,074158	0,975745
xbmc	11,815173	3 515,588855	0,991776
zookeeper	0,395452	197,310951	0,979616

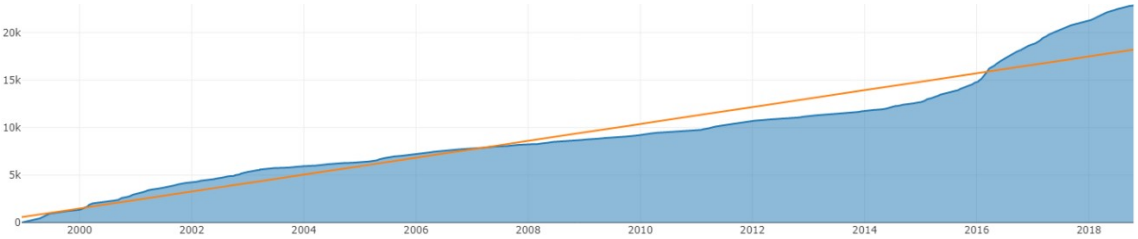
Kumulált sorok számához számított lineáris regressziós értékek:

Repository neve	Kumulált sorok száma esetén vett lineáris regresszió meredeksége	Kumulált sorok száma esetén vett lineáris regresszió metszéspontja	Kumulált sorok száma esetén vett lineáris regresszió R-négyzet értéke
angular	2 562,354627	-163 390,953721	0,993772
atom	1 501,916946	1 140 828,989006	0,852274
bootstrap	904,478766	41 376,032659	0,976490
brew	143,929986	-76 792,451690	0,876482
chef	482,876336	-123 296,419011	0,986562
coreclr	15 797,151514	4 012 775,579614	0,893690
corefx	10 977,455764	1 025 710,297185	0,955560
cpython	2 116,480533	-3 231 131,357675	0,952731
django	2 212,407540	-696 561,252042	0,893777
drupal	1 426,065261	-2 077 878,215888	0,806494
ember.js	570,821797	128 271,652012	0,990002
ghc	893,336070	973 775,566263	0,992589
hadoop	2 399,436365	6 302,416930	0,987060
homebrew-core	412,596089	-210 645,653299	0,936381
jenkins	494,087376	87 600,963337	0,986033
kendo-ui-core	2 281,476118	-409 113,991579	0,950043
mapbox-gl-js	909,982169	-63 114,843707	0,933409
mongo	6 880,958166	-4 850 492,413743	0,862410
moodle	5 161,559024	-2 586 118,712510	0,975905
neo4j	4 300,561464	-2 172 453,078958	0,969742
node	7 395,132405	-2 430 762,323190	0,962173
notepad-plus-plus	489,143028	290 096,026608	0,949275
opencv	4 403,289200	2 066 506,884902	0,944316
openssl	419,164751	351 964,745807	0,813506
openwrt	7 950,672098	-3 084 220,517779	0,991895
pandas	848,072754	-200 397,449852	0,987261
plotly.js	6 084,736190	-3 323 115,207045	0,804030
postgres	2 194,441366	737 327,051974	0,984879
rails	896,204875	-20 719,057972	0,943002
react	857,205069	90 690,731825	0,935161
roslyn	8 854,349331	3 201 320,785901	0,980650
scala	927,303904	-70 043,423124	0,985238
spark	6 122,606166	4 581 418,618646	0,650689
swagger-ui	766,340940	-311 938,274237	0,906409
TypeScript	14 024,124013	-568 264,166373	0,982417
vlc	7 752,633984	-6 866 764,800524	0,969818
vscode	5 813,845782	1 264 064,598337	0,985987
vue	842,523305	104 337,450008	0,918447
WordPress	686,710230	-438 963,266273	0,971766
xbmc	6 710,876790	13 205 941,712694	0,889585
zookeeper	134,576362	201 609,657587	0,831629

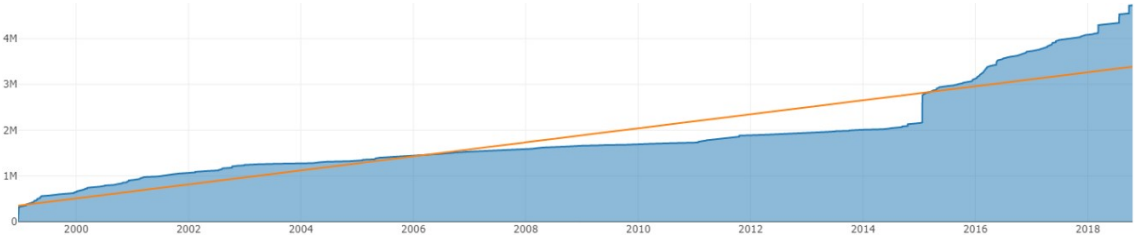
5.3.2.1 C

5.3.2.1.1 openssl

5-19. kép: openssl



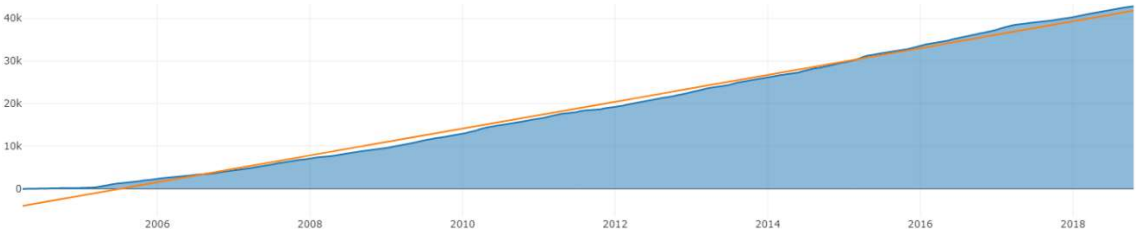
5-20. kép: openssl (kumulált commitok)



5-21. kép: openssl (kumulált sorok)

5.3.2.1.2 openwrt

5-22. kép: openwrt

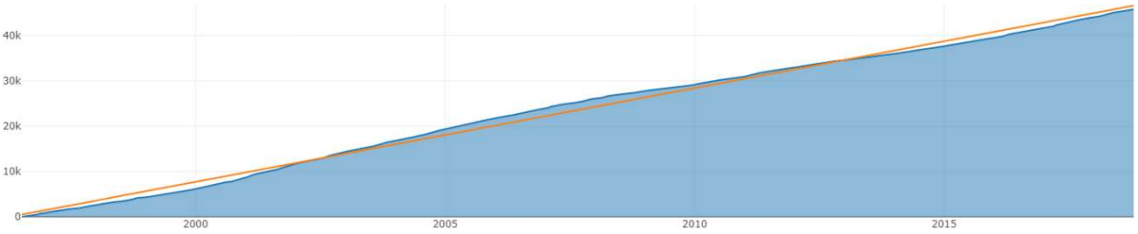


5-23. kép: openwrt (kumulált commitok)

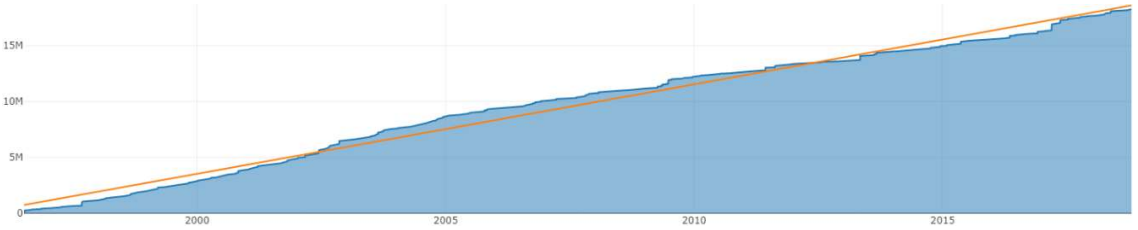
5-24. kép: openwrt (kumulált sorok)

5.3.2.1.3 postgres

5-25. kép: postgres



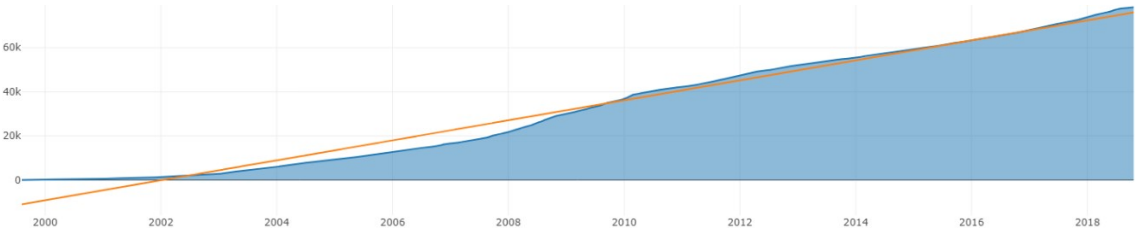
5-26. kép: postgres (kumulált commitok)



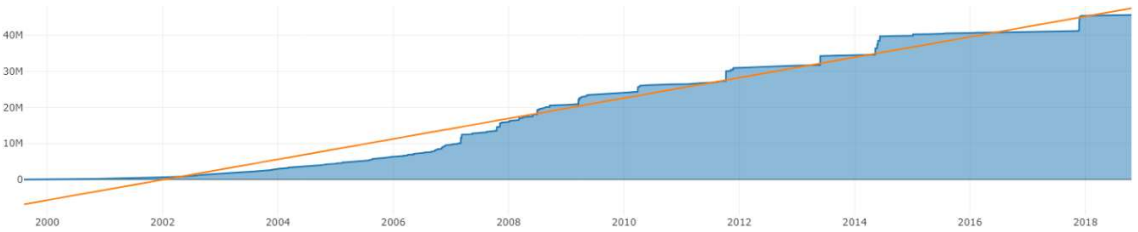
5-27. kép: postgres (kumulált sorok)

5.3.2.1.4 vlc

5-28. kép: vlc



5-29. kép: vlc (kumulált commitok)

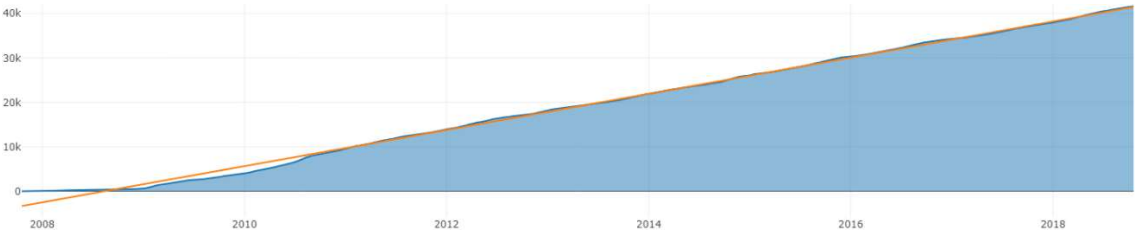


5-30. kép: vlc (kumulált sorok)

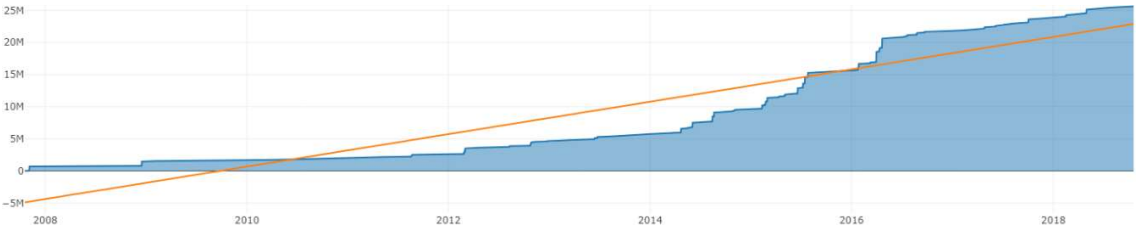
5.3.2.2 C++

5.3.2.2.1 mongo

5-31. kép: mongo

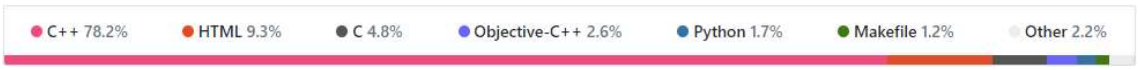


5-32. kép: mongo (kumulált commitok)

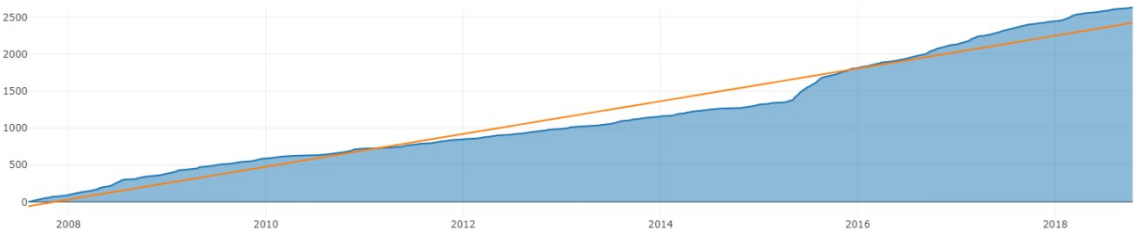


5-33. kép: mongo (kumulált sorok)

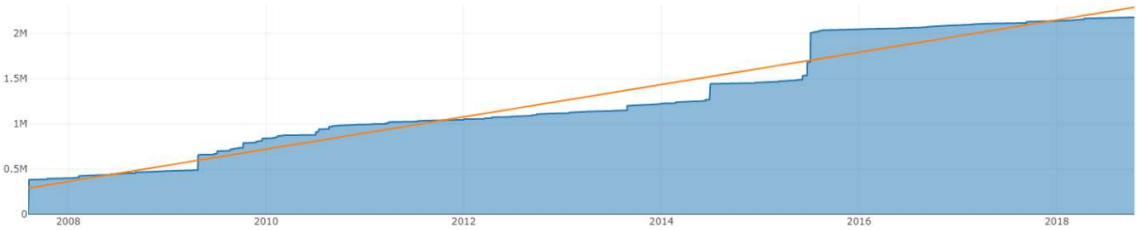
5.3.2.2.2 notepad-plus-plus



5-34. kép: notepad-plus-plus



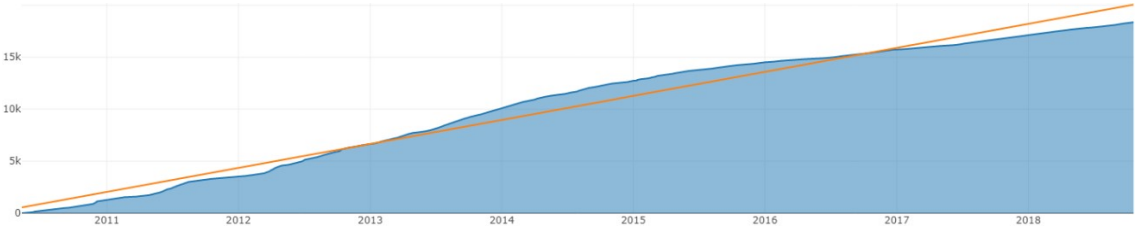
5-35. kép: notepad-plus-plus (kumulált commitok)



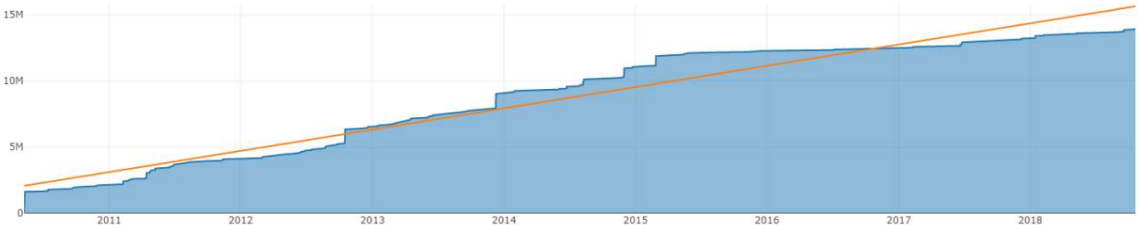
5-36. kép: notepad-plus-plus (kumulált sorok)

5.3.2.2.3 opencv

5-37. kép: opencv



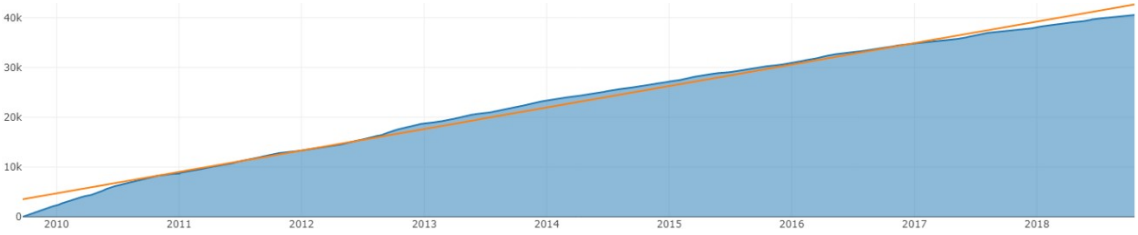
5-38. kép: opencv (kumulált commitok)



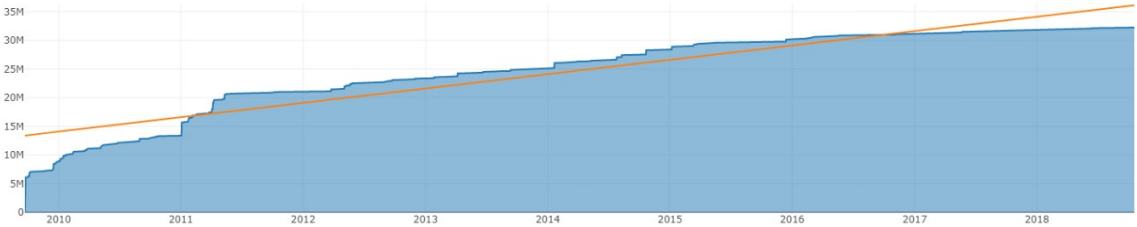
5-39. kép: opencv (kumulált sorok)

5.3.2.2.4 xbmc

5-40. kép: xbmc



5-41. kép: xbmc (kumulált commitok)

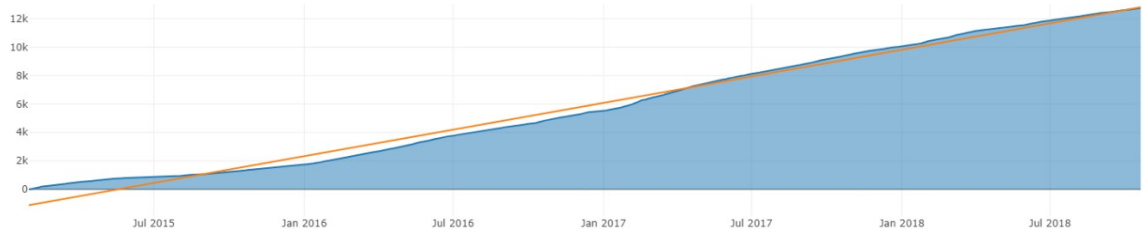


5-42. kép: xbmc (kumulált sorok)

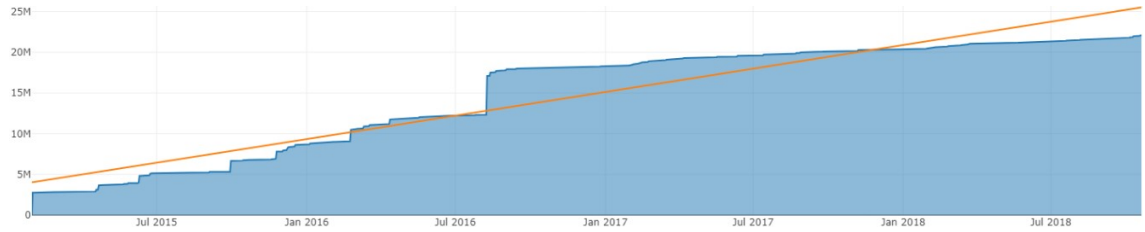
5.3.2.3 C#

5.3.2.3.1 coreclr

5-43. kép: coreclr



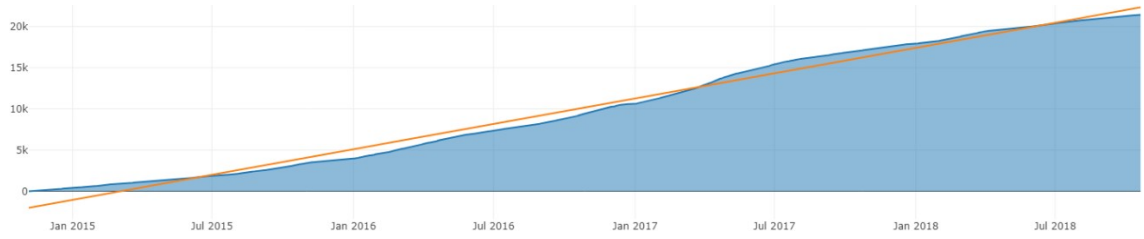
5-44. kép: coreclr (kumulált commitok)



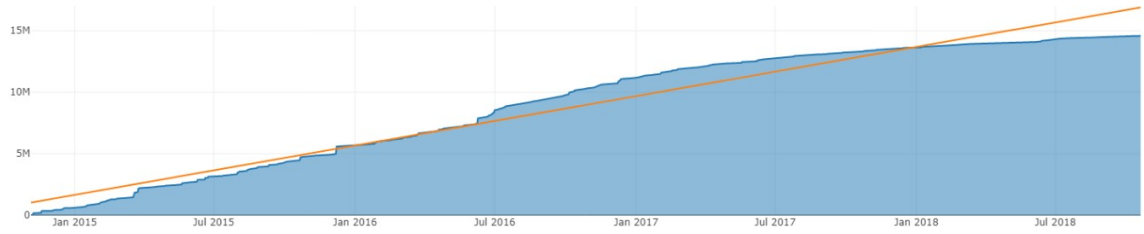
5-45. kép: coreclr (kumulált sorok)

5.3.2.3.2 corefx

5-46. kép: corefx



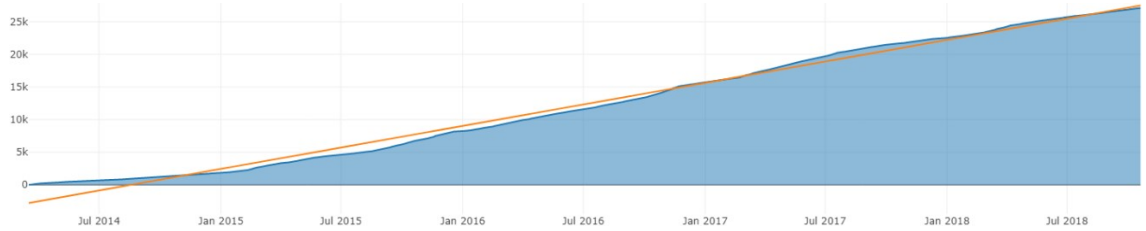
5-47. kép: corefx (kumulált commitok)



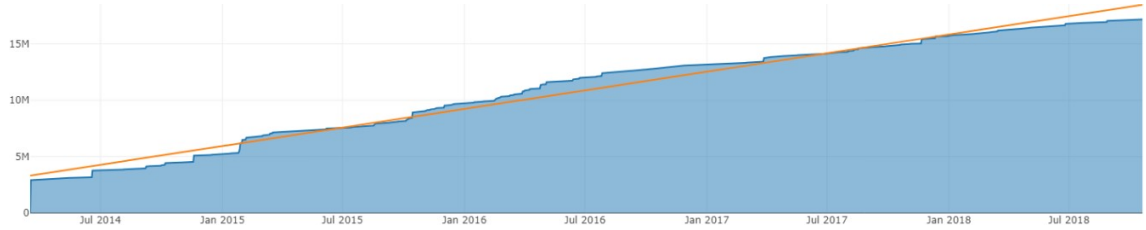
5-48. kép: corefx (kumulált sorok)

5.3.2.3.3 roslyn

5-49. kép: roslyn



5-50. kép: roslyn (kumulált commitok)

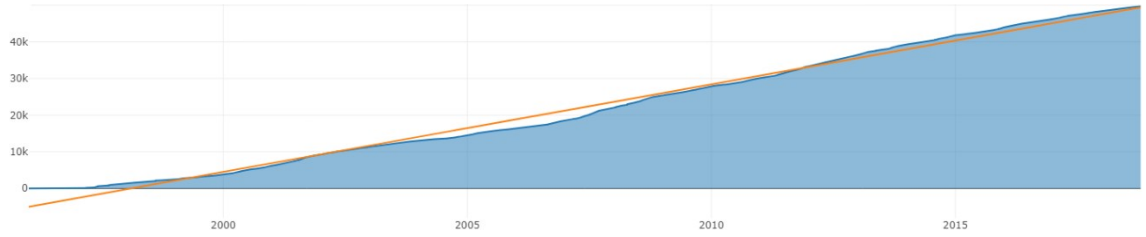


5-51. kép: roslyn (kumulált sorok)

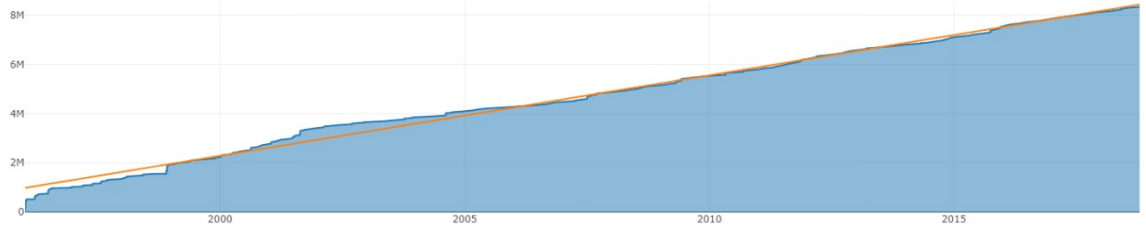
5.3.2.4 Haskell

5.3.2.4.1 ghc

5-52. kép: ghc



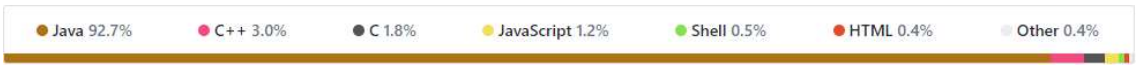
5-53. kép: ghc (kumulált commitok)



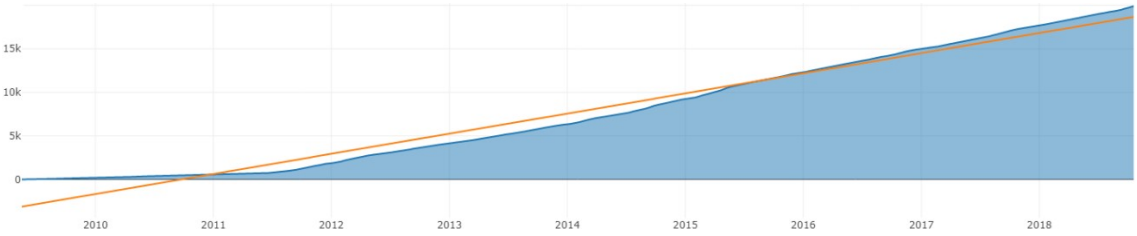
5-54. kép: ghc (kumulált sorok)

5.3.2.5 Java

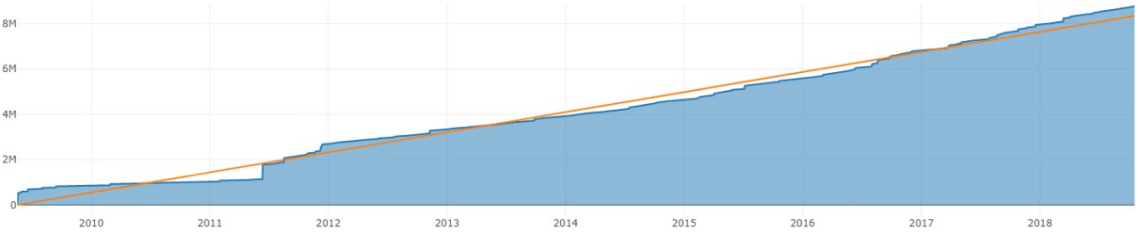
5.3.2.5.1 hadoop



5-55. kép: hadoop



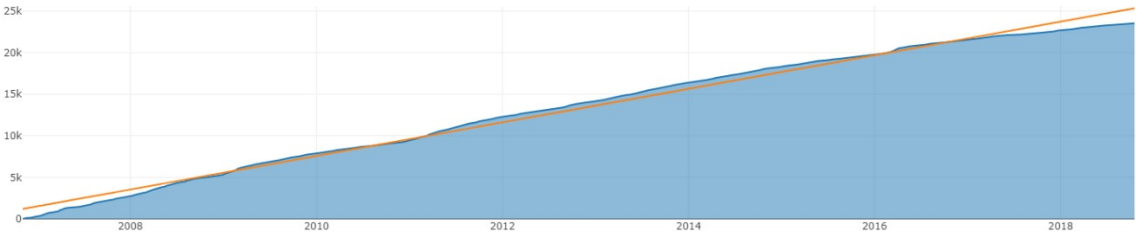
5-56. kép: hadoop (kumulált commitok)



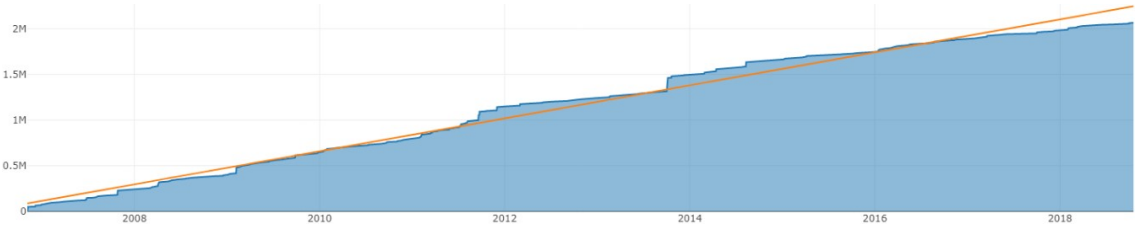
5-57. kép: hadoop (kumulált sorok)

5.3.2.5.2 jenkins

5-58. kép: jenkins



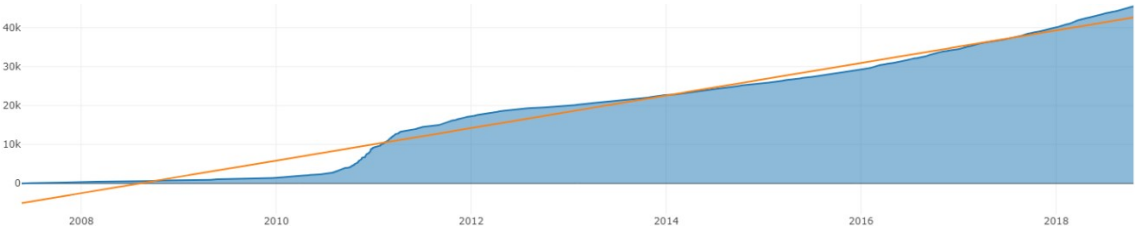
5-59. kép: jenkins (kumulált commitok)



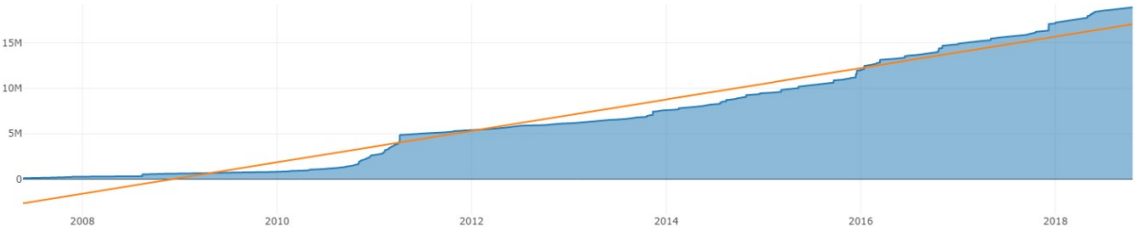
5-60. kép: jenkins (kumulált sorok)

5.3.2.5.3 neo4j

5-61. kép: neo4j

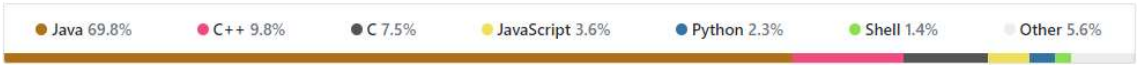


5-62. kép: neo4j (kumulált commitok)

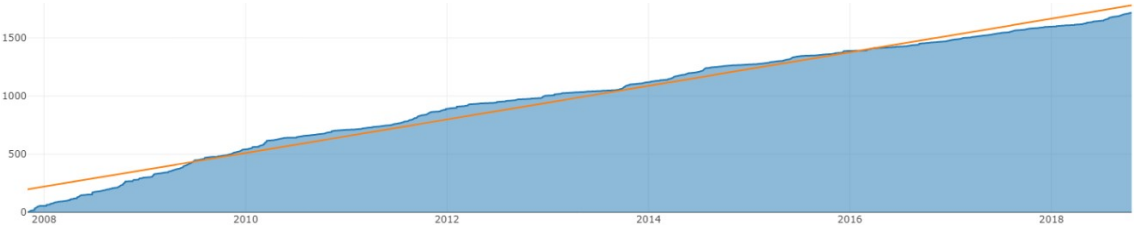


5-63. kép: neo4j (kumulált sorok)

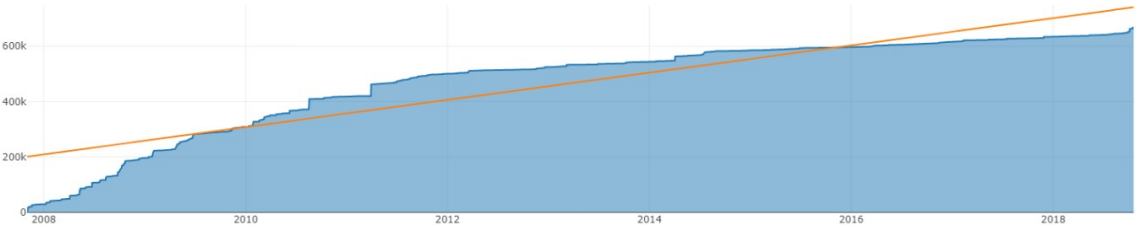
5.3.2.5.4 zookeeper



5-64. kép: zookeeper



5-65. kép: zookeeper (kumulált commitok)

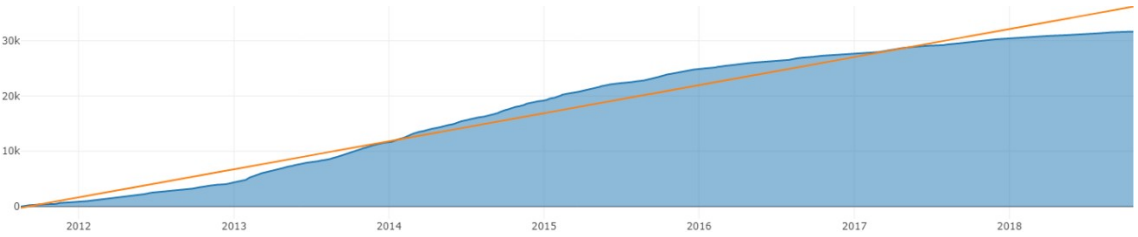


5-66. kép: zookeeper (kumulált sorok)

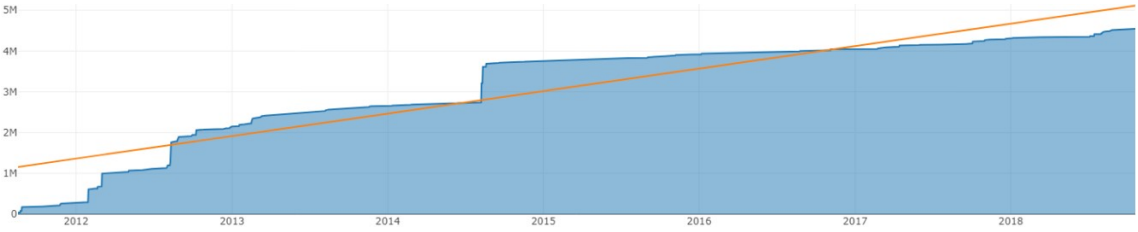
5.3.2.6 JavaScript

5.3.2.6.1 atom

5-67. kép: atom

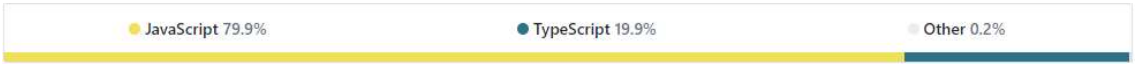


5-68. kép: atom (kumulált commitok)

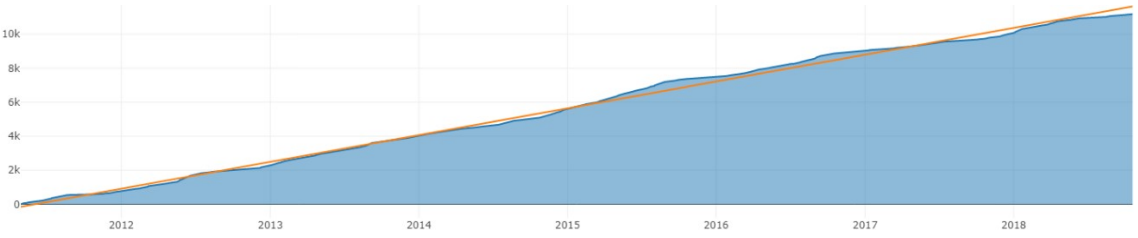


5-69. kép: atom (kumulált sorok)

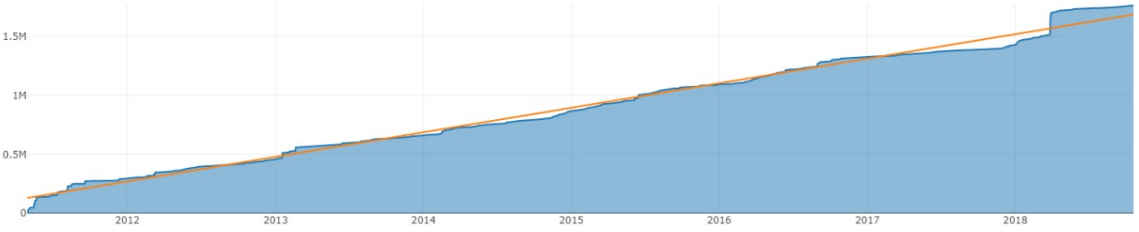
5.3.2.6.2 ember.js



5-70. kép: ember.js



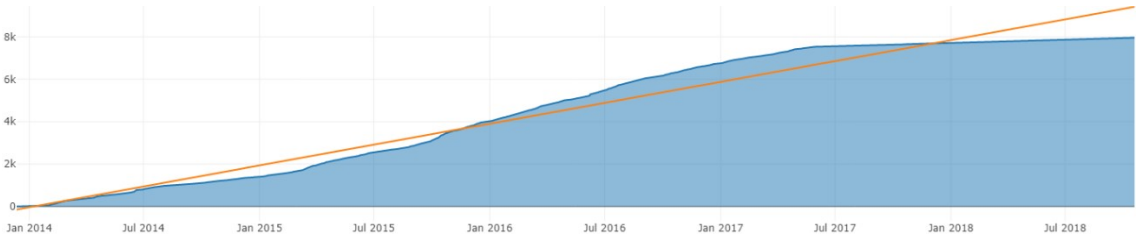
5-71. kép: ember.js (kumulált commitok)



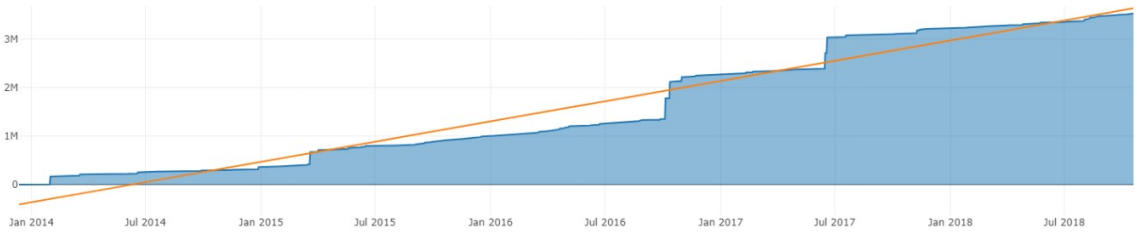
5-72. kép: ember.js (kumulált sorok)

5.3.2.6.3 kendo-ui-core

5-73. kép: kendo-ui-core



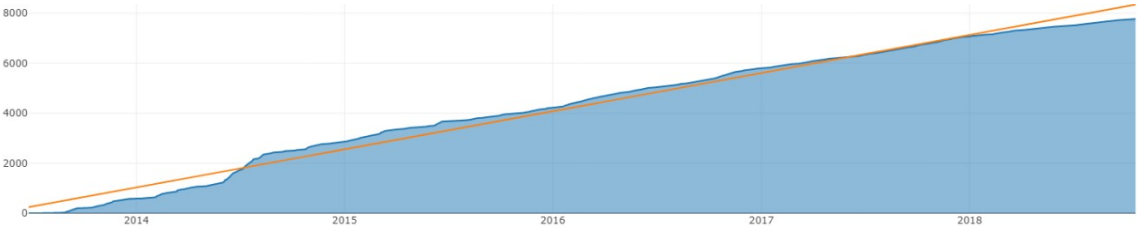
5-74. kép: kendo-ui-core (kumulált commitok)



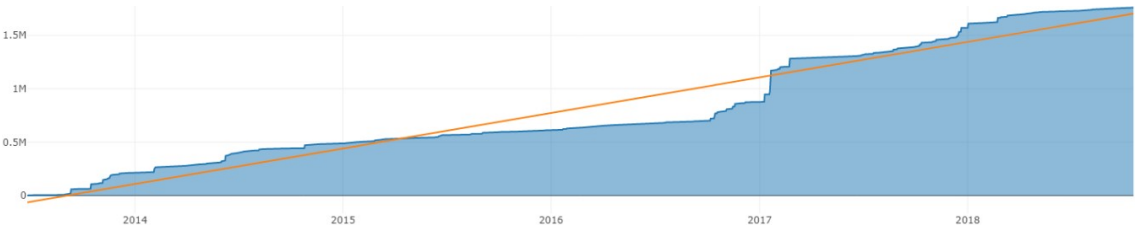
5-75. kép: kendo-ui-core (kumulált sorok)

5.3.2.6.4 mapbox-gl-js

5-76. kép: mapbox-gl-js



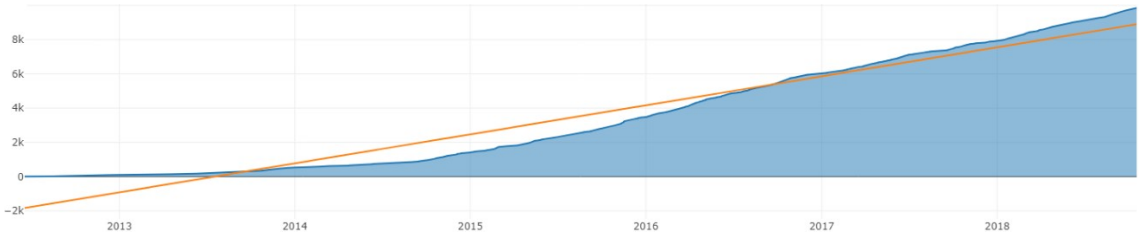
5-77. kép: mapbox-gl-js (kumulált commitok)



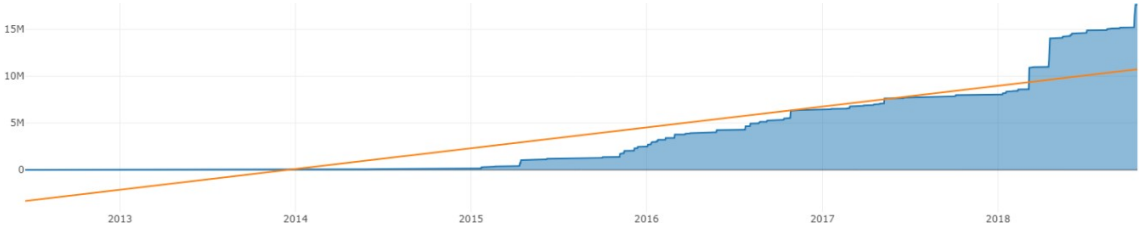
5-78. kép: mapbox-gl-js (kumulált sorok)

5.3.2.6.5 plotly.js

5-79. kép: plotly.js

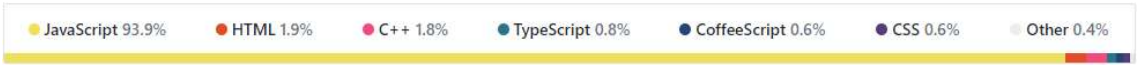


5-80. kép: plotly.js (kumulált commitok)

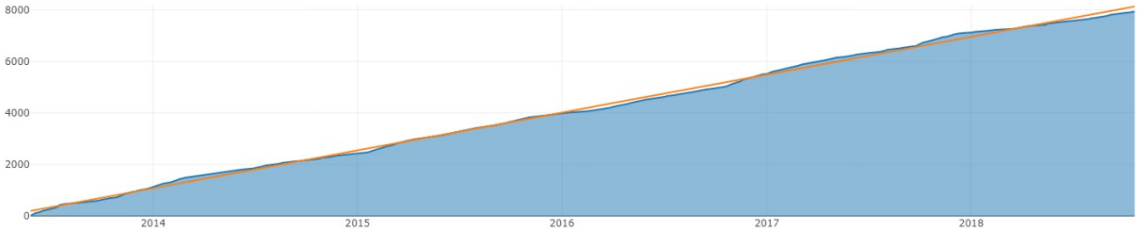


5-81. kép: plotly.js (kumulált sorok)

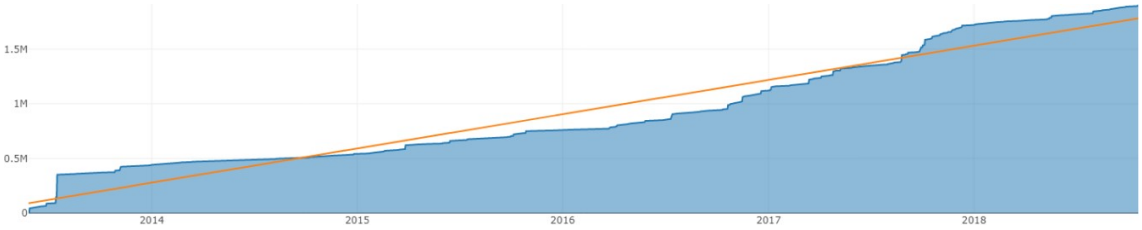
5.3.2.6.6 react



5-82. kép: react



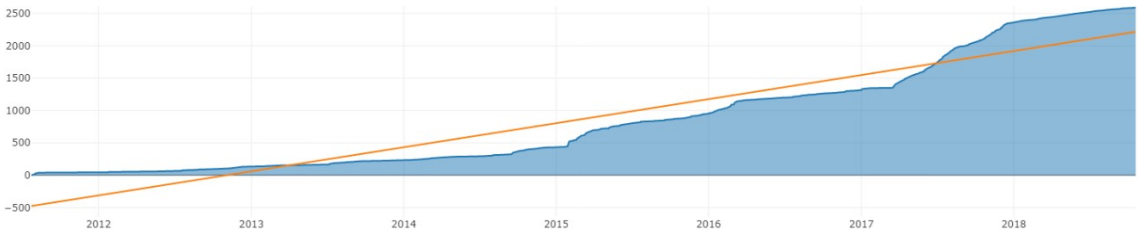
5-83. kép: react (kumulált commitok)



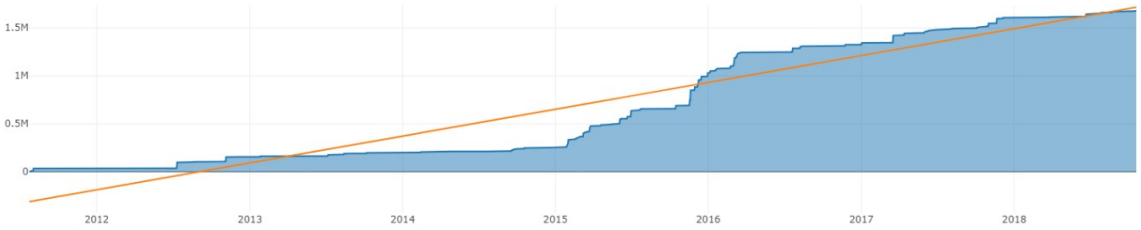
5-84. kép: react (kumulált sorok)

5.3.2.6.7 swagger-ui

5-85. kép: swagger-ui



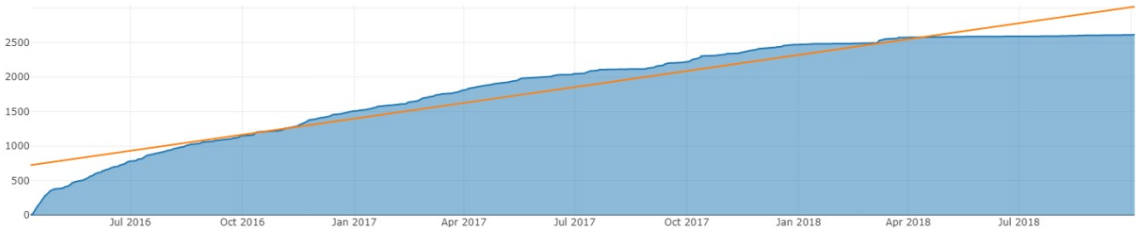
5-86. kép: swagger-ui (kumulált commitok)



5-87. kép: swagger-ui (kumulált sorok)

5.3.2.6.8 vue

5-88. kép: vue



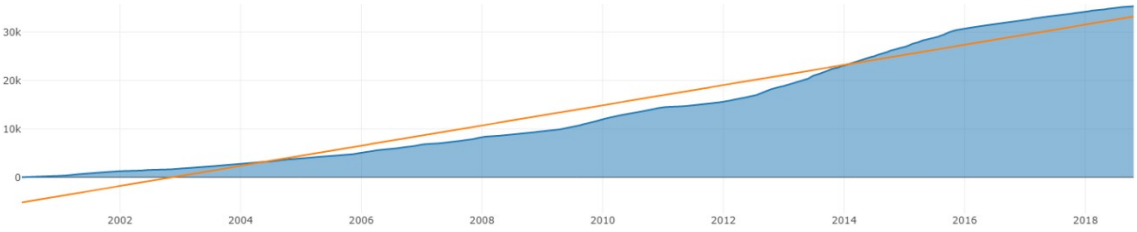
5-89. kép: vue (kumulált commitok)

5-90. kép: vue (kumulált sorok)

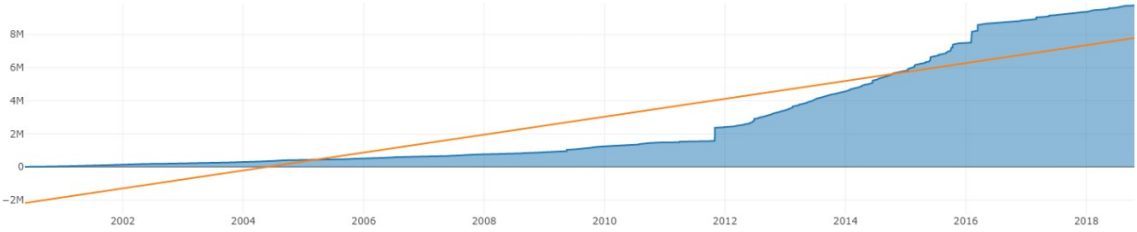
5.3.2.7 PHP

5.3.2.7.1 drupal

5-91. kép: drupal



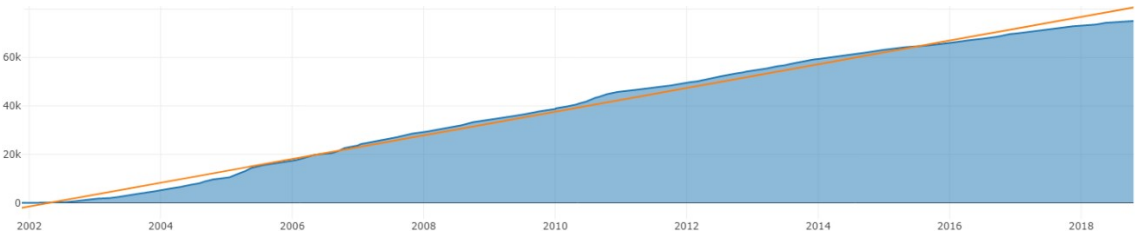
5-92. kép: drupal (kumulált commitok)



5-93. kép: drupal (kumulált sorok)

5.3.2.7.2 moodle

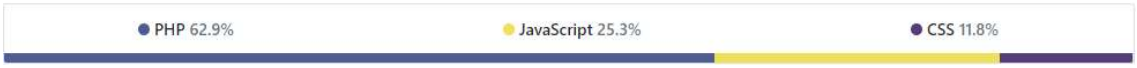
5-94. kép: moodle



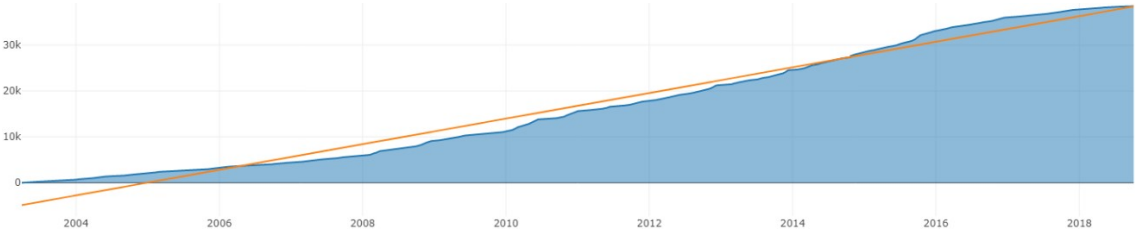
5-95. kép: moodle (kumulált commitok)

5-96. kép: moodle (kumulált sorok)

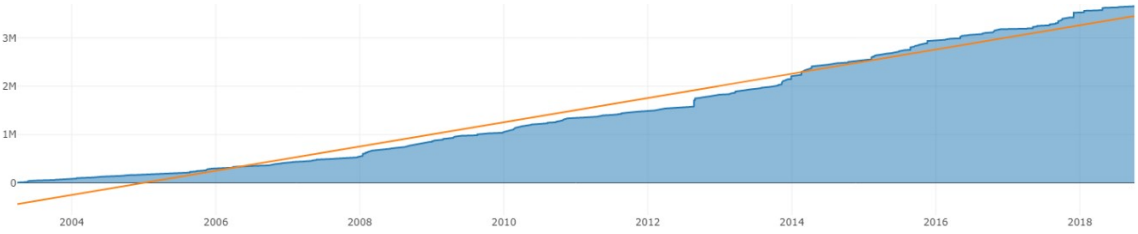
5.3.2.7.3 WordPress



5-97. kép: WordPress



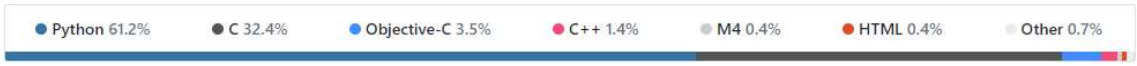
5-98. kép: WordPress (kumulált commitok)



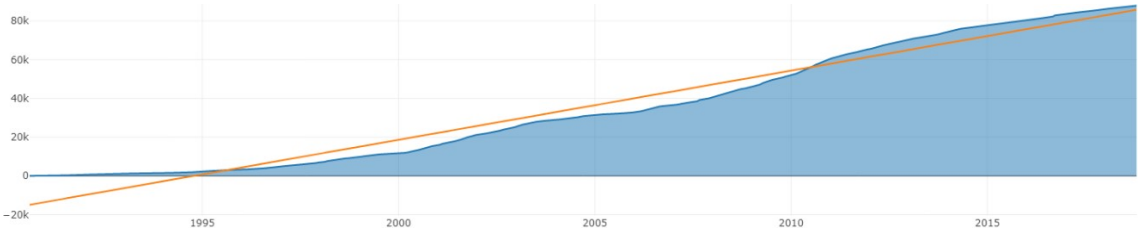
5-99. kép: WordPress (kumulált sorok)

5.3.2.8 Python

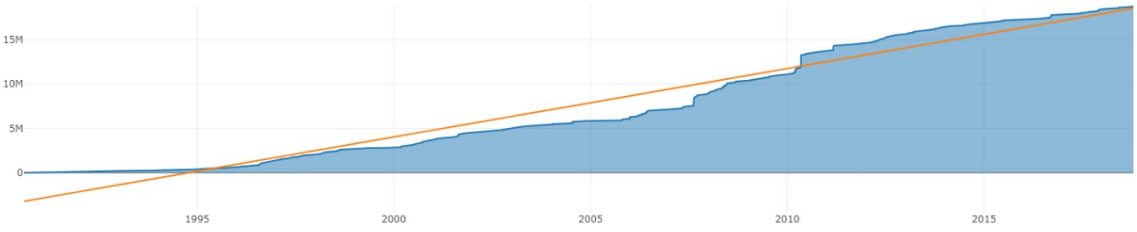
5.3.2.8.1 cpython



5-100. kép: cpython



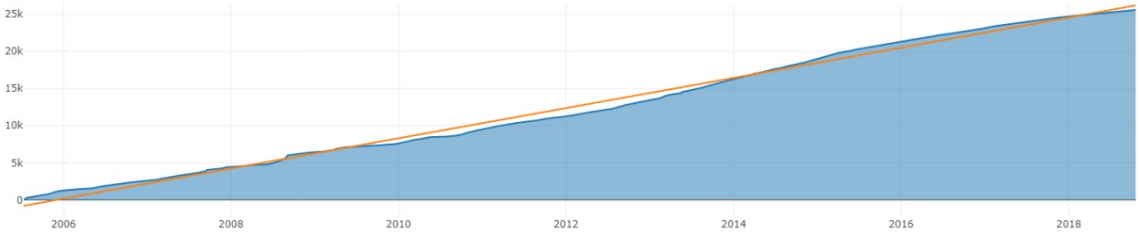
5-101. kép: cpython (kumulált commitok)



5-102. kép: cpython (kumulált sorok)

5.3.2.8.2 django

5-103. kép: django

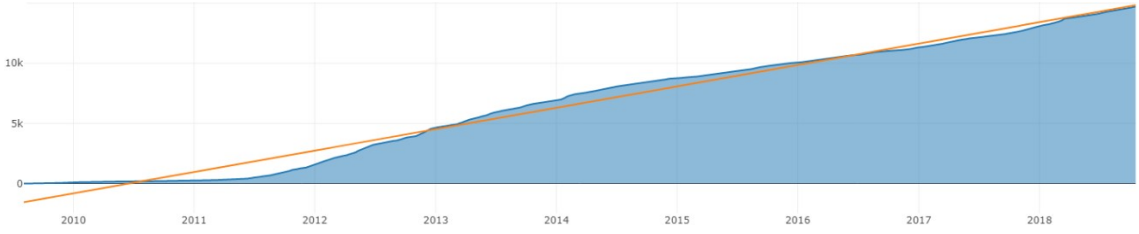


5-104. kép: django (kumulált commitok)

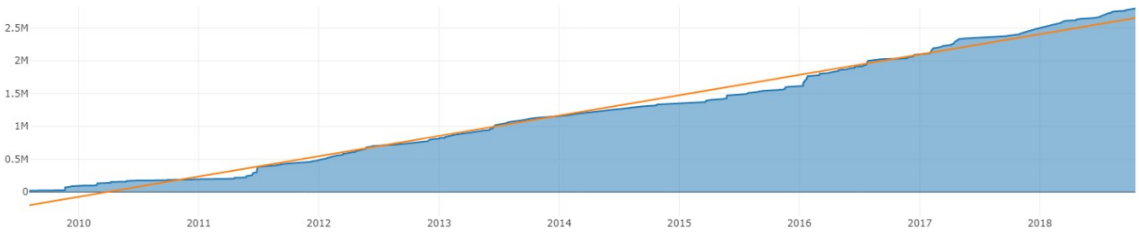
5-105. kép: django (kumulált sorok)

5.3.2.8.3 pandas

5-106. kép: pandas



5-107. kép: pandas (kumulált commitok)

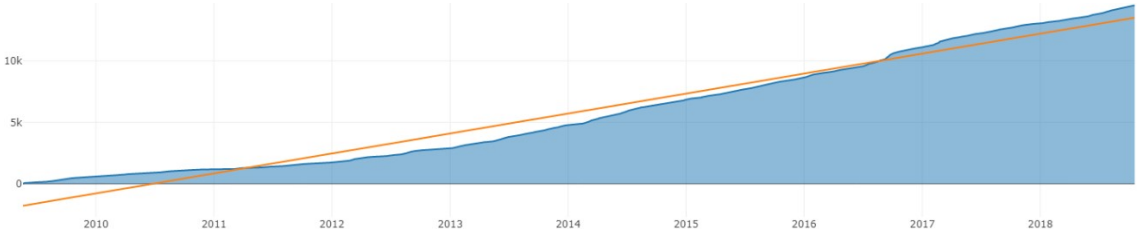


5-108. kép: pandas (kumulált sorok)

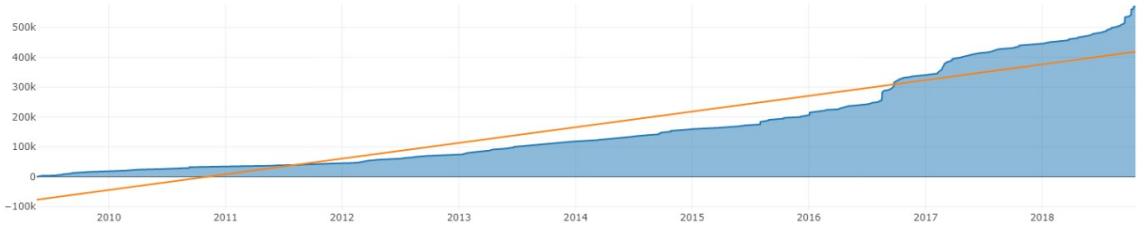
5.3.2.9 Ruby

5.3.2.9.1 brew

5-109. kép: brew



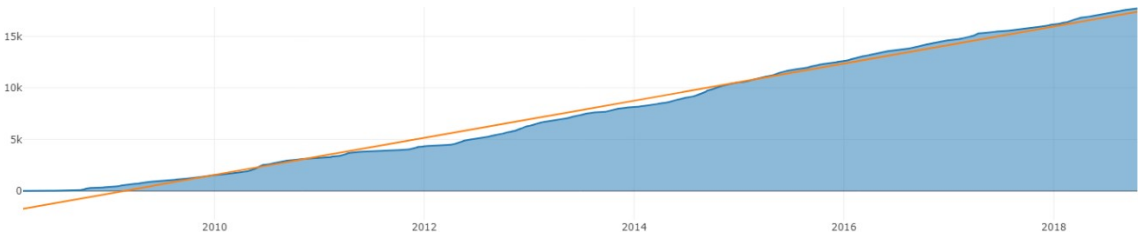
5-110. kép: brew (kumulált commitok)



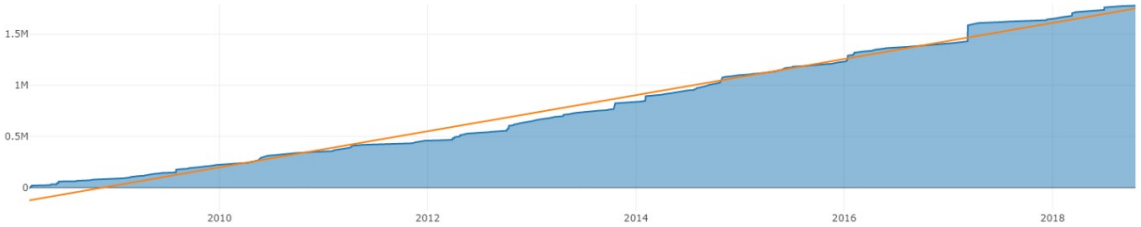
5-111. kép: brew (kumulált sorok)

5.3.2.9.2 chef

5-112. kép: chef



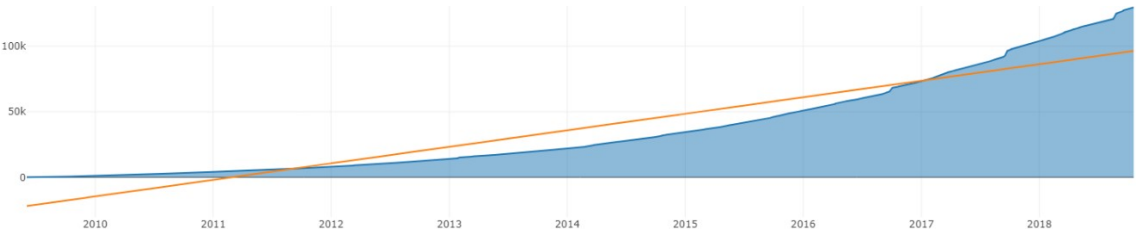
5-113. kép: chef (kumulált commitok)



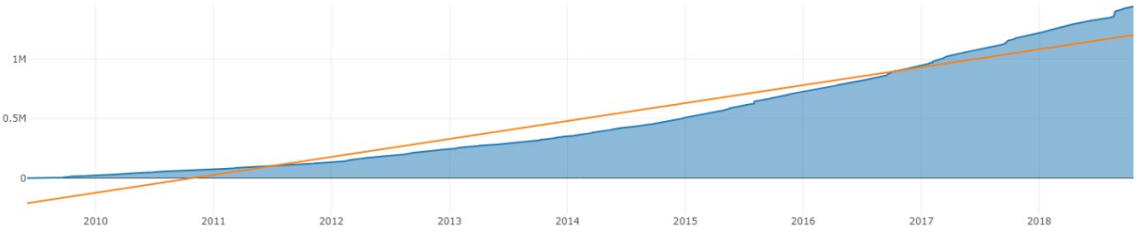
5-114. kép: chef (kumulált sorok)

5.3.2.9.3 homebrew-core

5-115. kép: homebrew-core



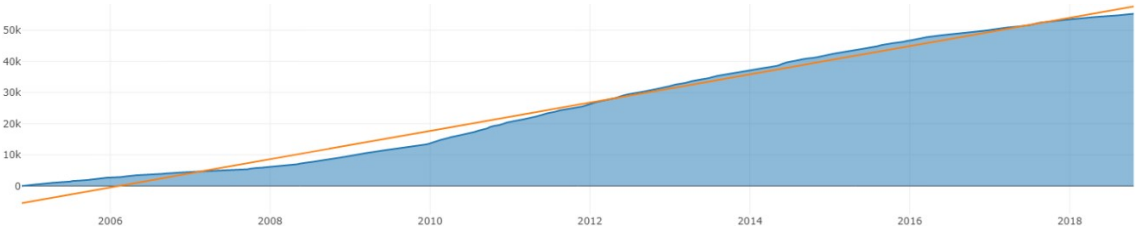
5-116. kép: homebrew-core (kumulált commitok)



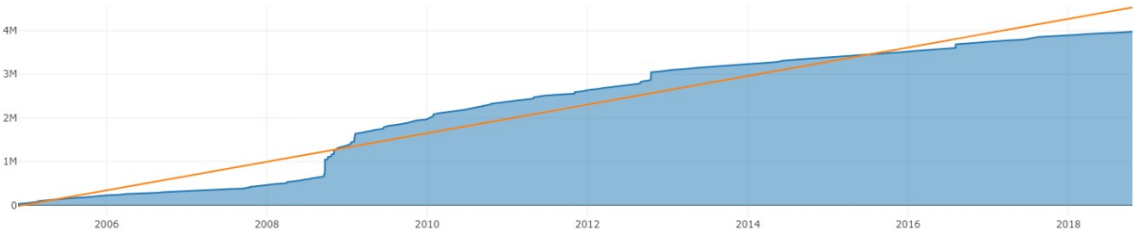
5-117. kép: homebrew-core (kumulált sorok)

5.3.2.9.4 rails

5-118. kép: rails



5-119. kép: rails (kumulált commitok)

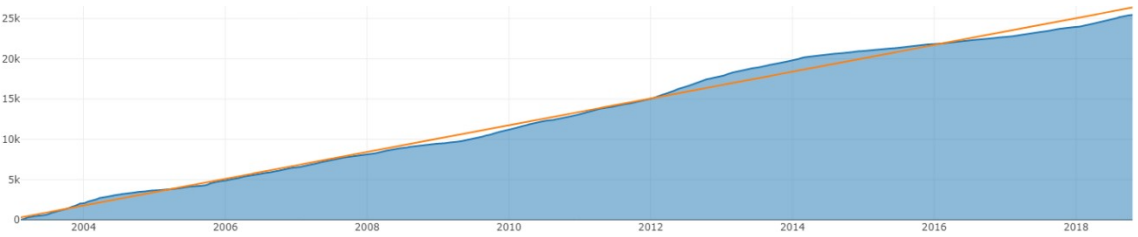


5-120. kép: rails (kumulált sorok)

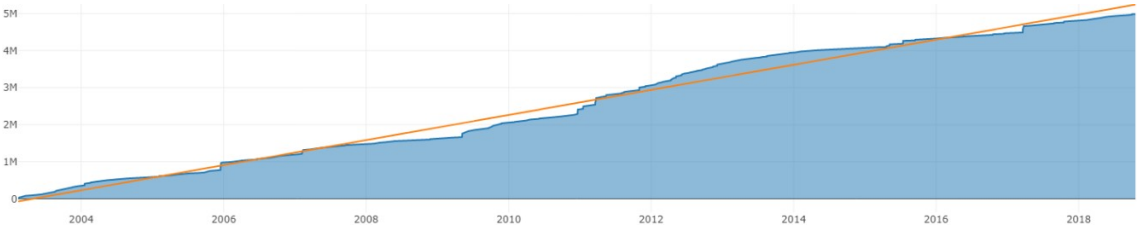
5.3.2.10 Scala

5.3.2.10.1 scala

5-121. kép: scala



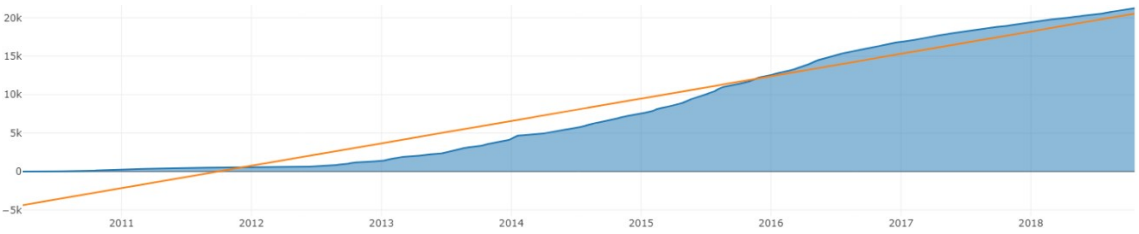
5-122. kép: scala (kumulált commitok)



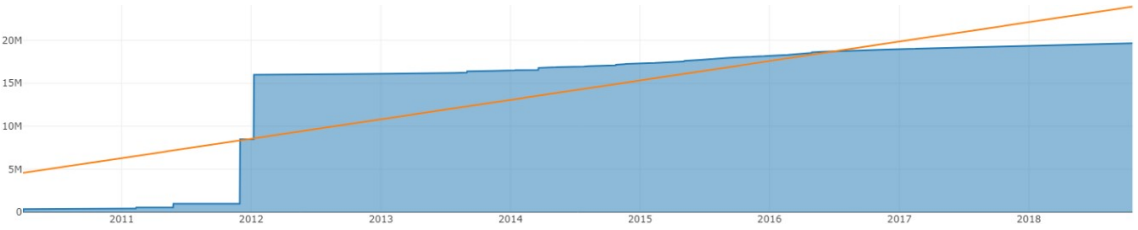
5-123. kép: scala (kumulált sorok)

5.3.2.10.2 spark

5-124. kép: spark



5-125. kép: spark (kumulált commitok)

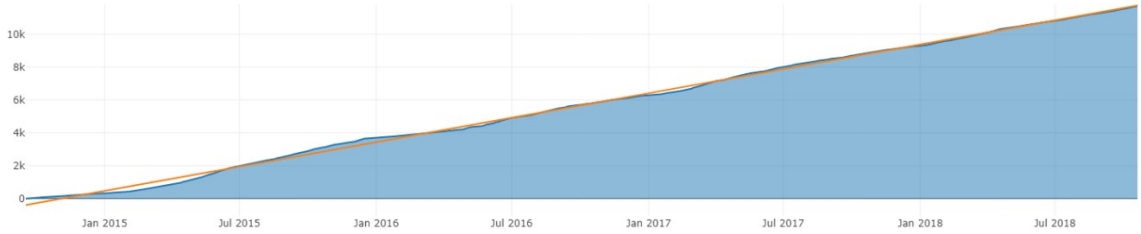


5-126. kép: spark (kumulált sorok)

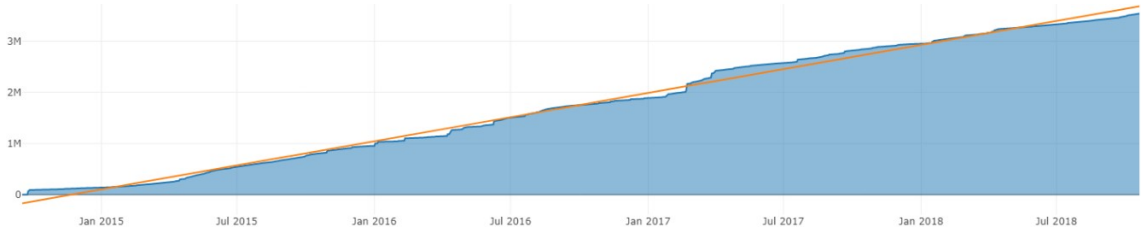
5.3.2.11 TypeScript

5.3.2.11.1 angular

5-127. kép: angular

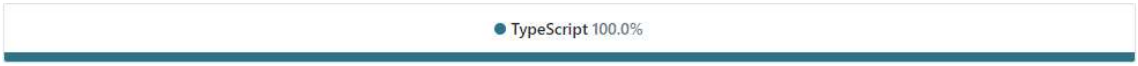


5-128. kép: angular (kumulált commitok)



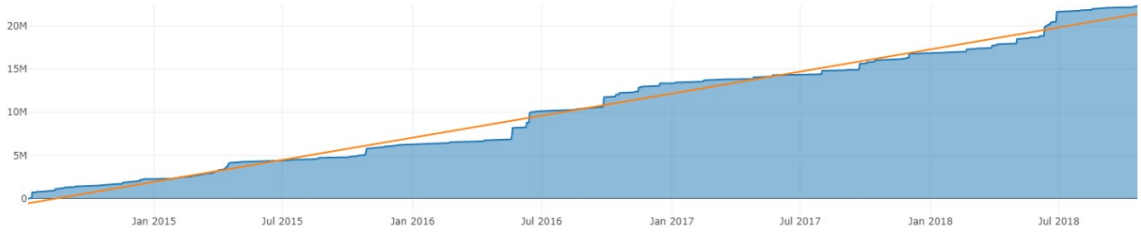
5-129. kép: angular (kumulált sorok)

5.3.2.11.2 TypeScript



5-130. kép: TypeScript

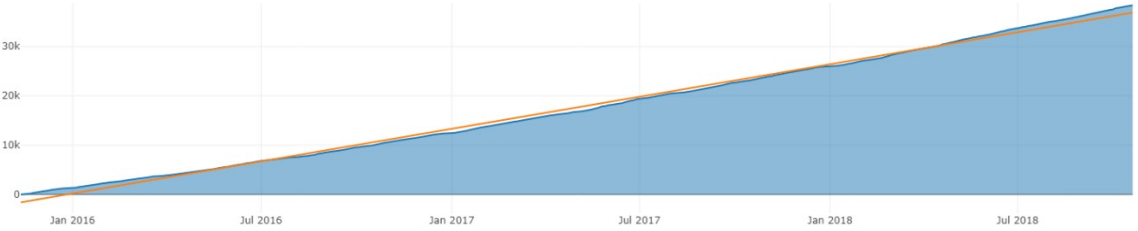
5-131. kép: TypeScript (kumulált commitok)



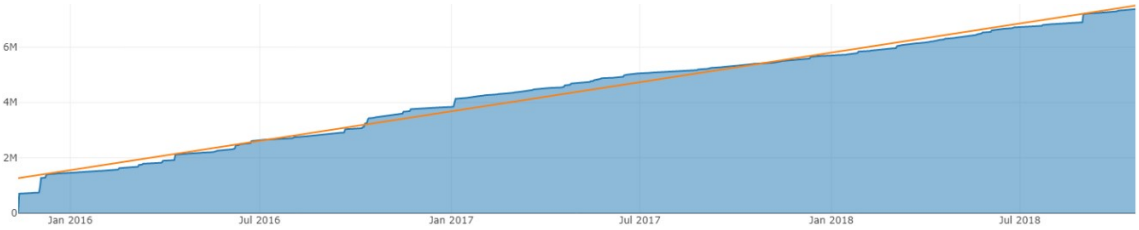
5-132. kép: TypeScript (kumulált sorok)

5.3.2.11.3 vscode

5-133. kép: vscode



5-134. kép: vscode (kumulált commitok)

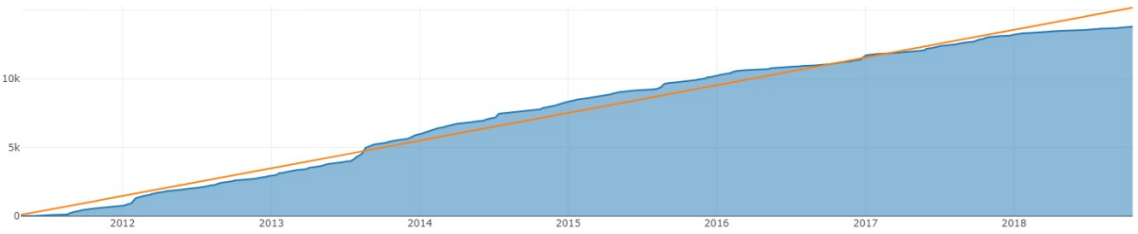


5-135. kép: vscode (kumulált sorok)

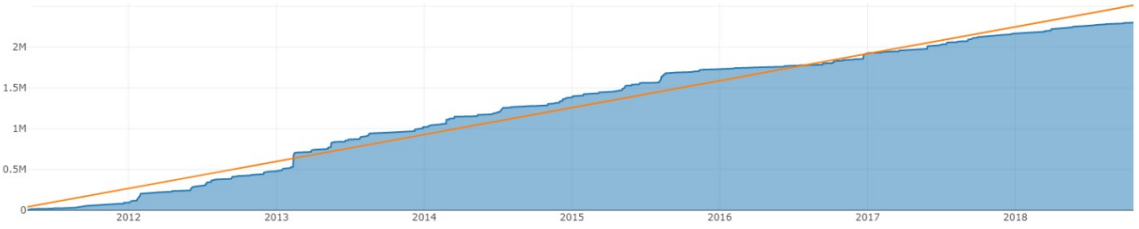
5.3.2.12 Vegyes

5.3.2.12.1 bootstrap

5-136. kép: bootstrap



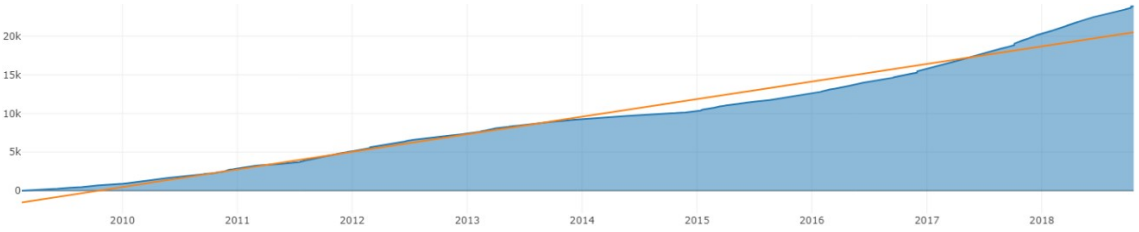
5-137. kép: bootstrap (kumulált commitok)



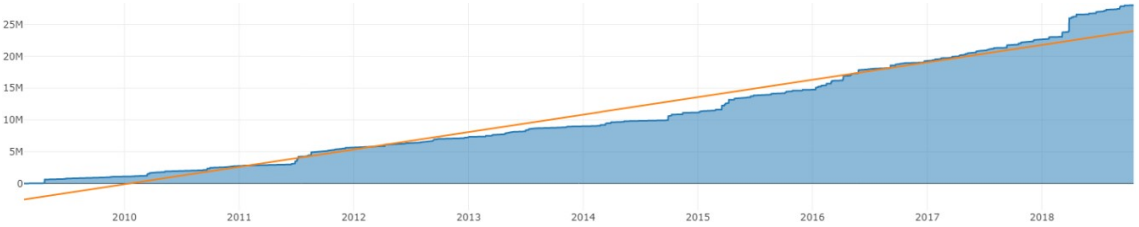
5-138. kép: bootstrap (kumulált sorok)

5.3.2.12.2 node

5-139. kép: node



5-140. kép: node (kumulált commitok)



5-141. kép: node (kumulált sorok)

5.3.2.13 Kis és közepes méretű GitHub repositoryk konklúziója

Az eredmények láttán az alábbi kategóriákat lehet megfigyelni:

angular
chef
coreclr
corefx
django
ember.js
ghc
jenkins
mapbox-gl-js
mongo
amoodle
openwrt
pandas
postgres
rails
react
roslyn
scala
TypeScript
vscode
xbmc

A részletezett 41 repository közül a legnagyobb kategóriát a kumulált commit számának diagram képe alapján szinte teljesen lineárisnak tekinthető repositoryk alkotják. Ezt alátámasztják az igen magas, 0,98-nál is nagyobb, R-négyzet értékek.

atom
brew
hadoop
kendo-ui-core
node
notepad-plus-plus
opencv
openssl
plotly.js
vlc
WordPress
zookeeper

A második legnagyobb kategóriát azok a repositoryk alkotják, amelyeknél a kumulált commitok számának diagramja hosszútávon lineáris szakaszokat mutat, és az ezek között lévő törések sem számukban, sem mértékükben nem jelentősek. Általában 0,95 és 0,98 közti R-négyzet érték jellemzi őket.

A kendo-ui-core esetében további észrevétel, hogy 2017 közepéig teljesen lineárisnak tekinthető, majd megtört az addigi trend és meredekség. A jelenséget tovább vizsgálva feltűnt, hogy az authorok száma az adott időponttól kezdődően lezuhant, és jellemzően már csak 1-2 név szerepel a historyban. Vélhetően 2017 közepétől már csak terméktámogatási vagy hibajavítási céllal történtek változások a repositoryban, a valódi fejlesztés egy másik verzióra vagy termékre fókuszált.

A plotly.js esetében egy töréspont két jól elkülöníthető szakaszra bontja a diagram képét. Mindkét szakasz önmagában teljesen lineáris, azonban a kezdeti időszakban kisebb meredekség, míg a projekt felfutását követően nagyobb meredekség jellemzi.

swagger-ui

Külön említést érdemel a swagger-ui, amelynél a diagram időintervallumának első felében teljesen lineáris volt, majd megtört ez a trend és kevésbé becsülhetővé vált. A második fél adatai 0,89-re rontották az R-négyzet értéket.

bootstrap
cpython
drupal
neo4j
spark

Kevésbé egyértelműen lineáris alakú, de lineárisan mégis egészen jól becsülhető a következő kategória. A jólbecsülhetőséget mutatja az is, hogy minden esetben 0,93-as feletti, sőt akár 0,98-as R-négyzet értékű repository is van ebben a kategóriában.

Külön említést érdemel a neo4j, ahol a kumulált commitos és a kumulált soros diagram képe mind a látható megugrás, mind a teljes összeképet tekintve igencsak hasonló.

A kumulált commitok és a kumulált sorok számának diagramját összevetve külön érdekességet mutat a spark, amelynél hirtelen több millió sor módosítás történt, miközben a commitok számában nem volt kiugrás. Tehát ezekben a kirívó esetekben az adott commitok a szokásos egy commiton belüli változtatások sokszorosát tartalmazták.

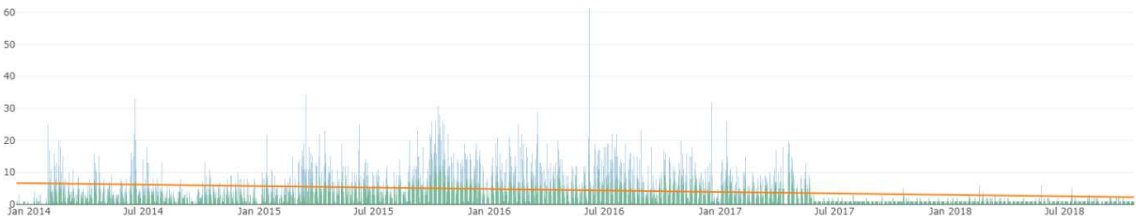
homebrew-core
vue

Az utolsó kategória esetén a legkevésbé jó a lineáris közelítés. Itt található az összes mért lineáris becslés közül a legrosszabb R-négyzet érték.

A homebrew-core a 0,87-es értékével a legrosszabbnak számít a részletesen bemutatott repositoryk lineáris becslései közül.

A másik, ránézésre a lineáris helyett inkább logaritmus függvény alakú, ezáltal a lineáris becslés szempontjából kevésbé jó repository, a vue. R-négyzet érték tekintetében ugyanakkor a 0,92 egész jó értéknek számít.

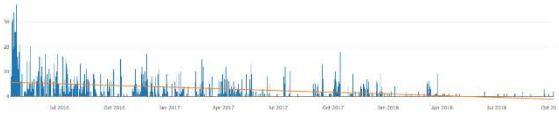
Két esetben nagyon valószínű, hogy a mért repositoryn az aktív fejlesztést időközben felváltotta a terméktámogató életciklus, amelynek keretében a projekten már főként csak hibajavítások történnek. Az egyik ilyen eset a kendo-ui-core, ahol jól láthatóan van egy töréspont a projekt fejlesztésében 2017 közepén.



5-142. kép: kendo-ui-core esetén a commitok (kékkel) és az authorok (zölddel) száma naponta

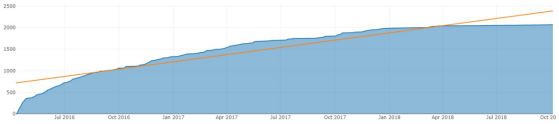
A másik esetben egyértelműen megállapítható, hogy a repositoryban történt fejlesztések túlnyomó többsége egy fejlesztő nevéhez köthetőek. A Vue.js repositoryjában kiemelkedő számokat produkált Evan You. Nem csak, hogy az ő nevéhez fűződik a legelső commit, hanem a projekt teljes időintervalluma alatt szinte teljes mértékben ő határozta meg a projekt fejlődését. Az alábbiakban összehasonlíthatóak az eredmények:

5-143. kép: vue (commitok)

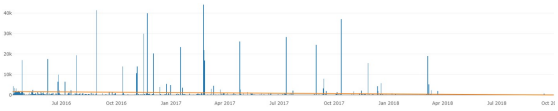


5-144. kép: Evan You (commitok)

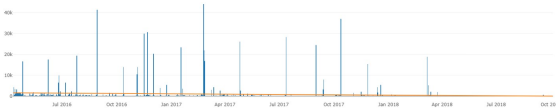
5-145. kép: vue (kumulált commitok)



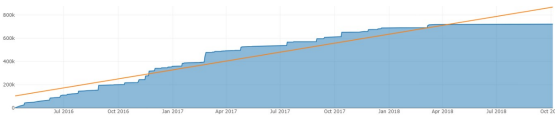
5-146. kép: Evan You (kumulált commitok)



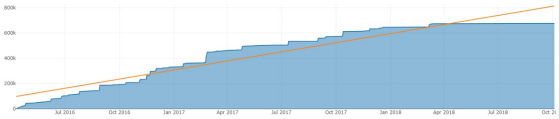
5-147. kép: vue (sorok)



5-148. kép: Evan You (sorok)



5-149. kép: vue (kumulált sorok)

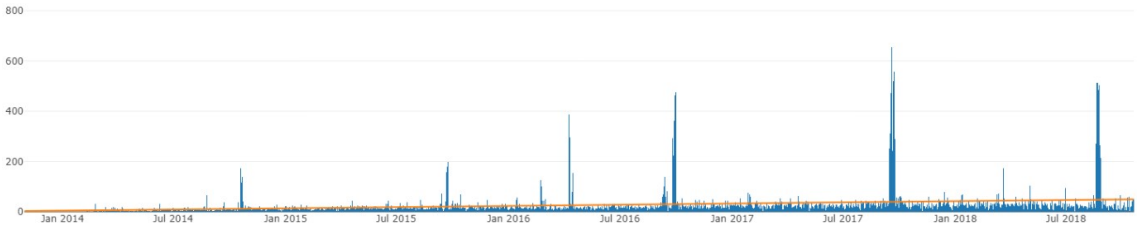


5-150. kép: Evan You (kumulált sorok)

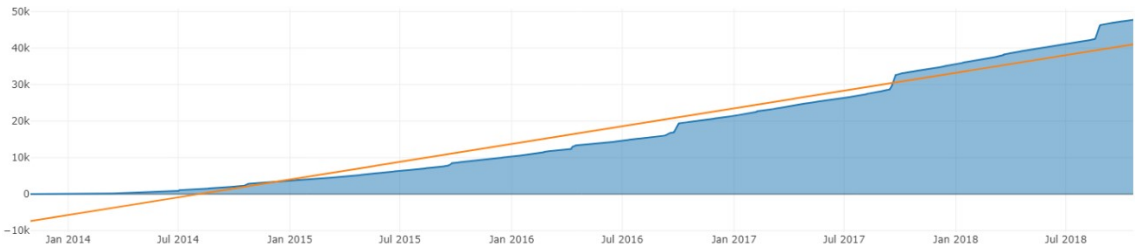
Amikor felhagyott az aktív munkával a repositoryn, akkor annak szinte meg is állt a fejlődése.

Pont ellenkező okból, tehát a lineárisnál is nagyobb ütemű commitszám növekedés miatt érdekes a homebrew-core repository. Felmerülhet a kérdés, hogy vajon ők tudják a többi projektnél is dinamikusabb ütemben fejlődés kulcsát? Nos, alaposabban utánajárva a válasz az, hogy nem. Csupán arról van szó, ami a Google esetében is történt. Időközben elkezdtek automatizmus által generált sorok is bekerülni a verziókezelő rendszerbe. Erre utal a commitolók közt szereplő BrewTestBot név is. Ez a név 2013.11.01. dátummal tűnik fel először. A kezdeti időszakban még napok teltek el a commitjai között (a leghosszabb commit nélküli időszaka 15 nap volt), majd gyakorlatilag egy-egy napnyi kiesést leszámítva mindennaposság váltak a commitjai.

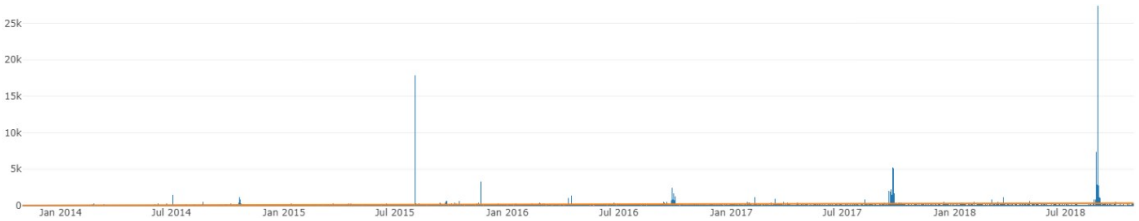
Először a jobb láthatóság érdekében nagyobb méretben szerepelnek majd a BrewTestBot felhasználónévhez tartozó előzmény adatok diagrammjai, majd egymás mellé téve megtekinthető lesz a látottak jelentősége a teljes repository szempontjából.



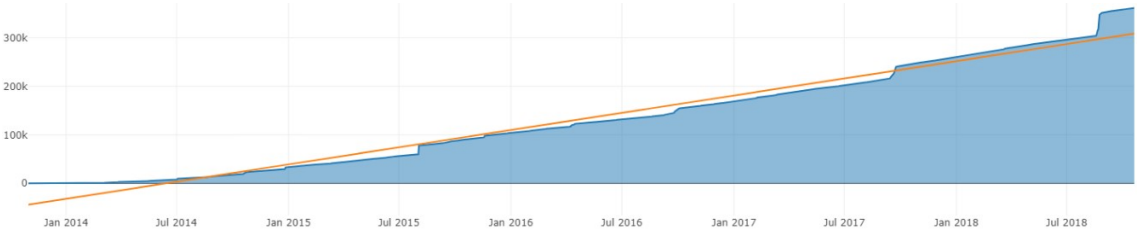
5-151. kép: BrewTestBot (commitok)



5-152. kép: BrewTestBot (kumulált commitok)



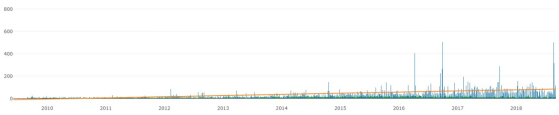
5-153. kép: BrewTestBot (sorok)



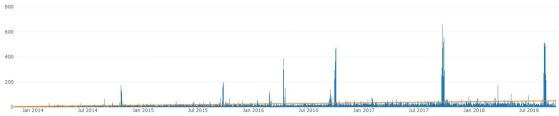
5-154. kép: BrewTestBot (kumulált sorok)

Az itt látható függvény alakokat, illetve a megugró értékeket kell keresni a repository ábráin is.

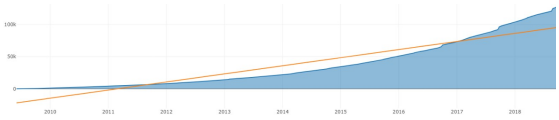
Az előző esethez hasonló, összehasonlító nézetben így néz ki a homebrew-core repository és a BrewTestBot nevű felhasználó kapcsolata:



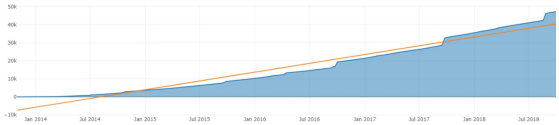
5-155. kép: homebrew-core (commitok)



5-156. kép: BrewTestBot (commitok)

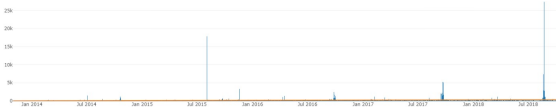


5-157. kép: homebrew-core (kumulált commitok)

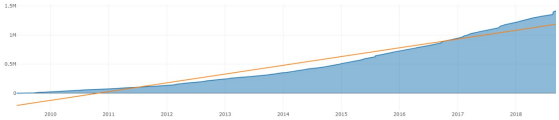


5-158. kép: BrewTestBot (kumulált commitok)

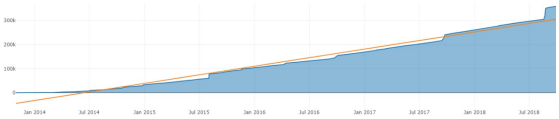
5-159. kép: homebrew-core (sorok)



5-160. kép: BrewTestBot (sorok)



5-161. kép: homebrew-core (kumulált sorok)



5-162. kép: BrewTestBot (kumulált sorok)

A repository esetében 2014 elejétől kezdenek túlmutatni az értékek a lineáris becslés egyenesén. Ez egybeesik a BrewTestBot commitok közti megjelenésével. A repository által a továbbiakban produkált eredmények nagyságrendileg egybeesnek az addigi adatok alapján becsült lineáris trend, valamint az adott időpont utáni generált, gépi commitok számainak összegével.

5.3.3 Nagyméretű GitHub repositoryk

Általános adatok táblázata:

Repository neve	Első és utolsó commit között eltelt napok száma	Commitos napok száma	Leghosszabb commit nélküli időszak hossza napokban	Commitok száma összesen	Sorok száma összesen
freebsd	9 261	9 178	11	251 488	324 778 029
gcc	10 923	9 548	263	164 738	70 788 649
linux	24 586	5 030	11 582	725 665	80 840 648
mysql-server	6 525	6 036	5	78 157	49 763 262
QGIS	5 950	4 956	59	47 159	77 604 593

Átlag értékek táblázata:

Repository neve	Átlag commit szám az összes napra vetítve	Átlag commit szám a commitos napokra vetítve	Átlag sorok száma az összes napra vetítve	Átlag sorok száma a commitos napokra vetítve	Átlag sorok száma egy commitra vetítve
freebsd	27,16	27,40	35 069,43	35 386,58	1 291,43
gcc	15,08	17,25	6 480,70	7 413,98	429,70
linux	29,52	144,27	3 288,08	16 071,70	111,40
mysql-server	11,98	12,95	7 626,55	8 244,41	636,71
QGIS	7,93	9,52	13 042,79	15 658,72	1 645,59

Kumulált commitok számához számított lineáris regressziós értékek:

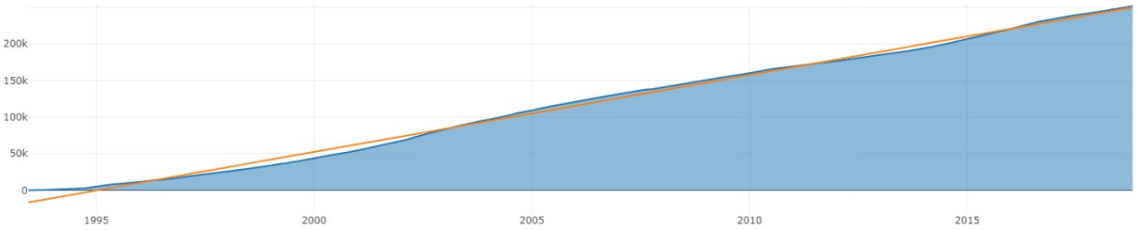
Repository neve	Kumulált commit számok esetén vett lineáris regresszió meredeksége	Kumulált commit számok esetén vett lineáris regresszió metszéspontja	Kumulált commit számok esetén vett lineáris regresszió R-négyzet értéke
freebsd	28,807584	-16 568,797629	0,995550
gcc	16,443331	-27 135,730170	0,967440
linux	40,297077	-234 202,578127	0,776873
mysql-server	13,086127	-6 624,766099	0,994766
QGIS	7,117451	-6 889,275033	0,904526

Kumulált sorok számához számított lineáris regressziós értékek:

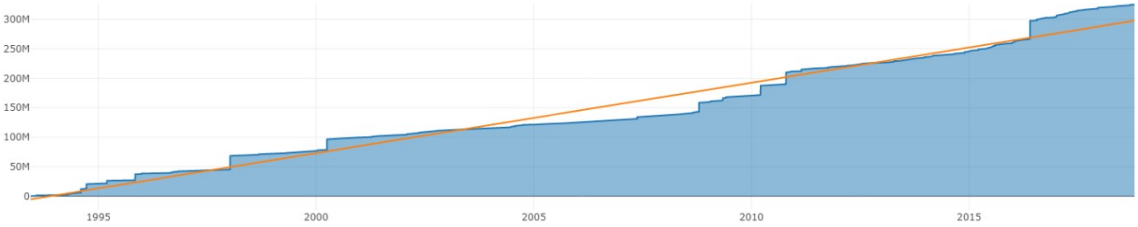
Repository neve	Kumulált sorok száma esetén vett lineáris regresszió meredeksége	Kumulált sorok száma esetén vett lineáris regresszió metszéspontja	Kumulált sorok száma esetén vett lineáris regresszió R-négyzet értéke
freebsd	32 702,002797	-5 345 710,309013	0,967792
gcc	7 092,704309	-14 906 942,880592	0,927222
linux	4 574,662350	-25 569 017,438549	0,802271
mysql-server	6 891,472777	-4 641 491,293340	0,964812
QGIS	13 372,453887	-15 233 583,225890	0,894535

5.3.3.1 *freebsd*

5-163. kép: freebsd



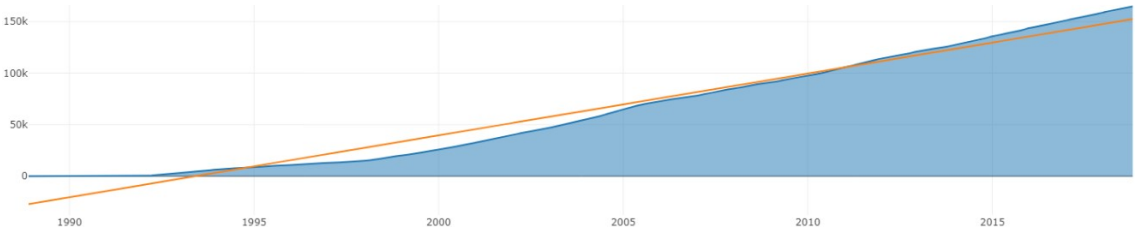
5-164. kép: freebsd (kumulált commitok)



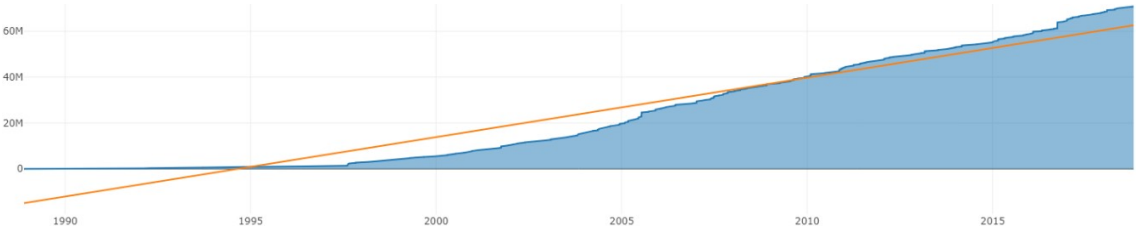
5-165. kép: freebsd (kumulált sorok)

5.3.3.2 *gcc*

5-166. kép: gcc



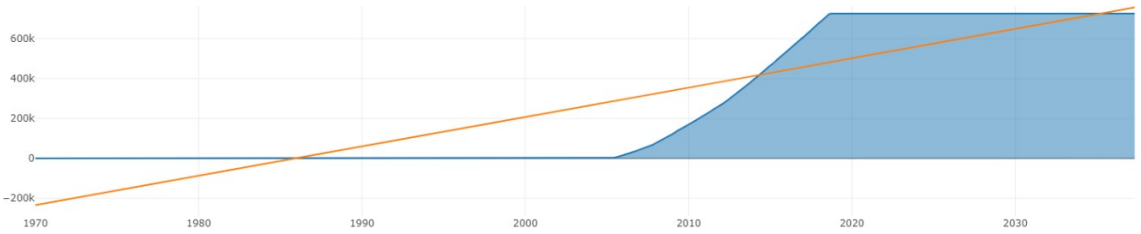
5-167. kép: gcc (kumulált commitok)



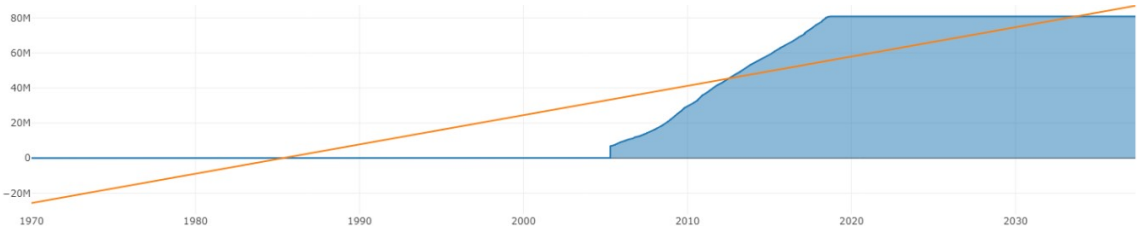
5-168. kép: gcc (kumulált sorok)

5.3.3.3 linux

5-169. kép: linux

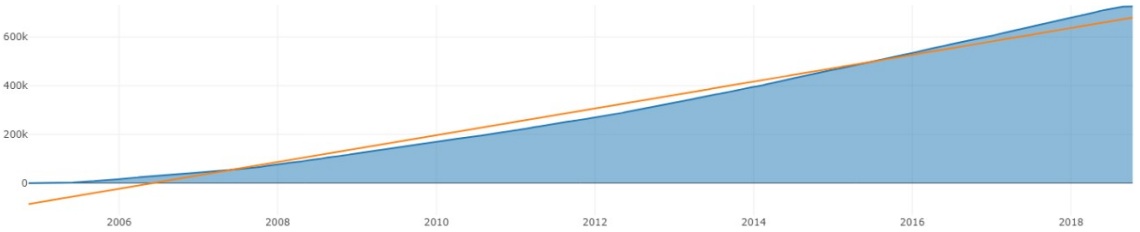


5-170. kép: linux (kumulált commitok)

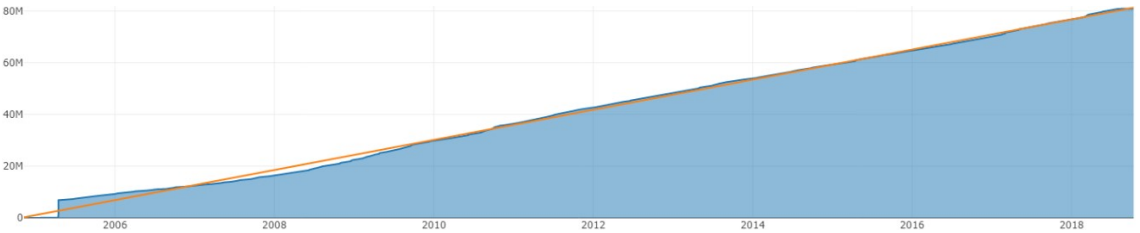


5-171. kép: linux (kumulált sorok)

A historyból kigyűjtött adatok, és ezáltal az azok alapján készült diagram is, vélhetően korrupt, hamis információkat is tartalmaz. Ennek következménye a fenti diagramokon is jól látszik. Három egymás utáni szakasz figyelhető meg. Az első a kirívóan múltba nyúlik, míg a harmadik a jövőidőbe. Az ilyesfajta szélsőértékek nélküli, feltehetően az eredetinél helyesebb, korrigált adatok viszont az alábbi diagramokat eredményezik:



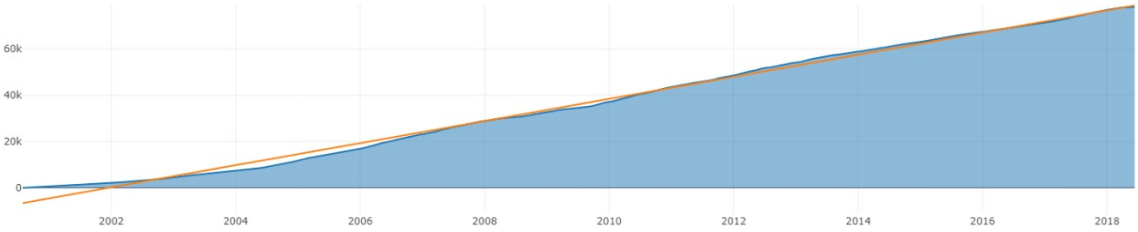
5-172. kép: linux (kumulált commitok) a korrigálás után



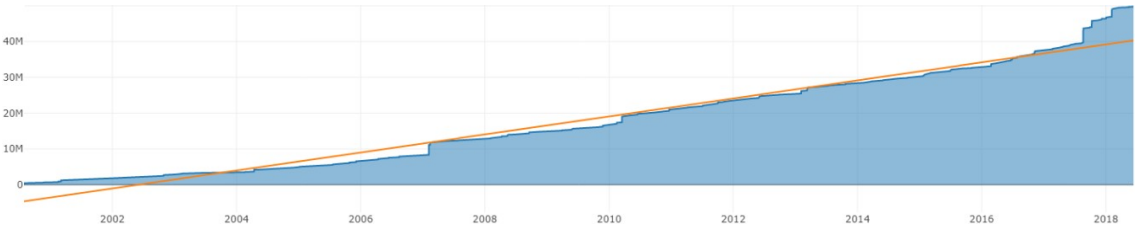
5-173. kép: linux (kumulált sorok) a korrigálás után

5.3.3.4 mysql-server

5-174. kép: mysql-server



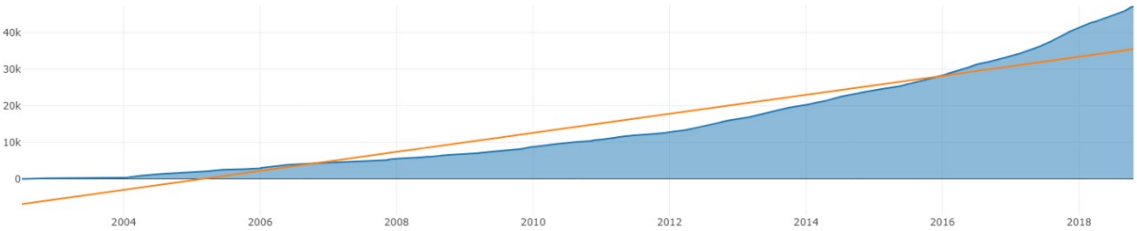
5-175. kép: mysql-server (kumulált commitok)



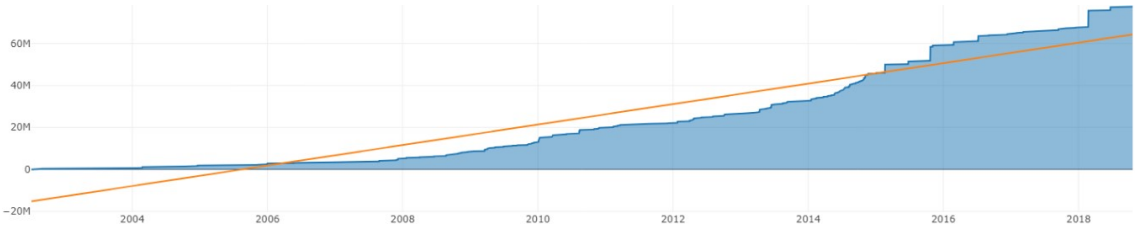
5-176. kép: mysql-server (kumulált sorok)

5.3.3.5 QGIS

5-177. kép: QGIS



5-178. kép: QGIS (kumulált commitok)



5-179. kép: QGIS (kumulált sorok)

5.3.3.6 Nagyméretű GitHub repositoryk konklúziója

Az eredmények láttán az alábbi kategóriákat lehet megfigyelni:

freebsd
mysql-server

A freebsd és a mysql-server esetén teljesen lineáris trend figyelhető meg. Mindkét esetben 0,99 feletti R-négyzet értékkel. Figyelembevéve a hosszú időintervallumot, valamint a commitok és változó sorok nagyságrendjét ez a két repository kiemelkedő példája a linearitásnak.

gcc

A gcc szintén teljesen lineáris trendet mutat, azzal a megjegyzéssel, hogy az 1998 körüli időszakban kimutathatóan megnőtt az egyenes meredeksége.

linux

A linux repository esetén a history alapján kigyűjtött adatok egy része vélhetően korrupt és érvénytelen. Ez magyarázhatja az időben kirívóan korai, és egyben gyanúsan Unix epoch dátumhoz (1970.01.01.) kapcsolódó, illetve a triviálisan lehetetlen, jövőbeli history bejegyzéseket (mint például a 2037.04.25). Ezek nagymértékben torzítják a diagramot.

Egy ilyesfajta torzítástól mentes eredményhez szükség volt az előzmény adatok feltételezett korrigálására. Ez jelen esetben annyit jelentett, hogy az egyértelműen hibás időpontú commitokat, valamint a szélsőértéknek számító néhány commitot nem vettem figyelembe a mérés és ezáltal az eredmény kiértékelése során. Az így kapott új diagram képe sokkal jobb eredményt adott a lineáris becsléssel. Számszerüsítve a korrigálás előtt a commitos mérés R-négyzet értéke 0,776873 volt, míg az elvégzett korrigálás után 0,980403 lett.

QGIS

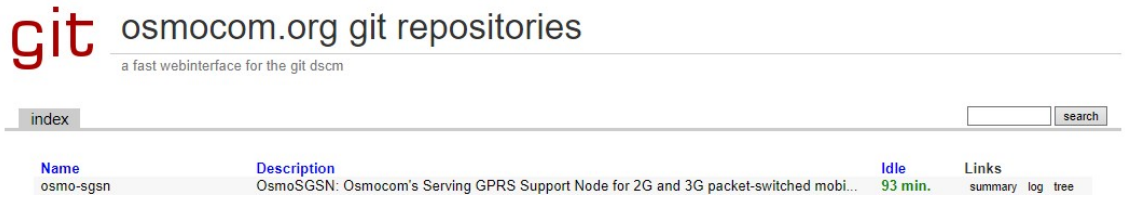
A QGIS a maga 0,9-es R-négyzet értékével a kevésbé lineárisan változó esetek közé tartozik.

5.3.4 Osmocom projekthez kapcsolódó repositoryk

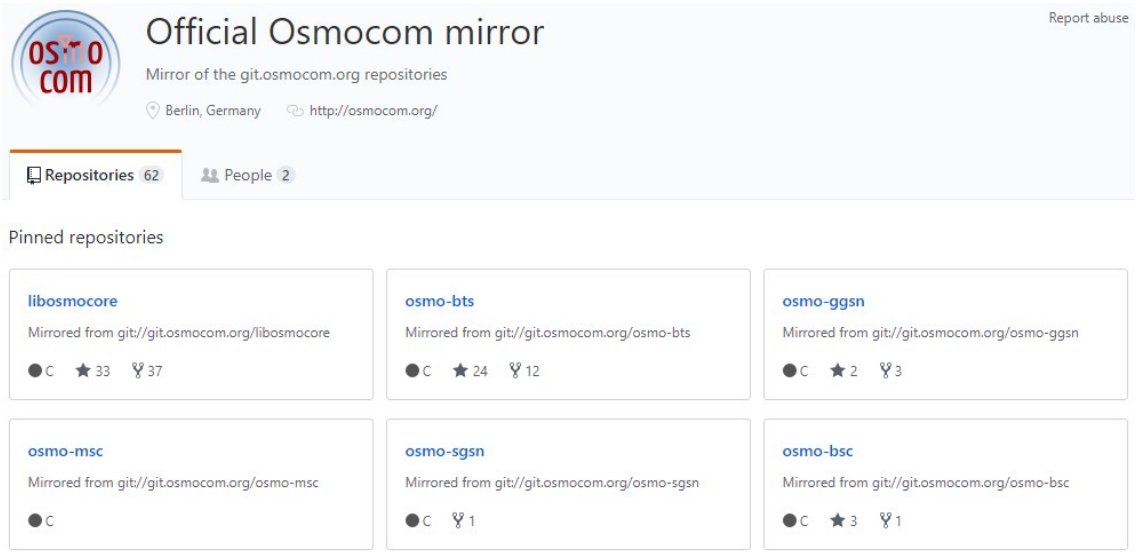
Az Osmocom (Open Source Mobile Communications) egy nyílt forráskódú mobil kommunikációval foglalkozó gyűjtőprojekt. Magába foglal különböző mobil kommunikációs sztenderdeket (beleértve a GSM, DECT, TETRA, stb.) implementáló szoftvereket és egyéb szoftveres eszközöket.

A hivatalos repositoryk a http://git.osmocom.org címen érhetőek el, ugyanakkor tükrözve megtalálhatóak a https://github.com/osmocom címen is.

A projekthez tartozó nagymennyiségű repositoryra való tekintettel, azoknak csak egy része kerül bemutatásra. A kiválasztott repositoryk csakis az előre ismerhető információk alapján lettek kiválasztva, tehát nem befolyásolta a döntést az eredmények ismerete. Szűrési feltételt jelentett a hivatalos oldalon feltüntetett aktuális tétlenségi idő, amely alapján az utóbbi egy hónapban aktív repositoryk halmazát adta, valamint további szűrési lehetőséget adott a GitHub oldalon kitűzött, kiemelt repositoryk listája. Jelen esetben a második feltétel pont az első egy részhalmazát jelentette, tehát a kiemelt repositoryk az elvártnak megfelelően a legaktívabban fejlesztett kódbázisok között voltak, ezekről részletes diagram is lesz az alábbiakban. [30] [31] [32]



5-180. kép: Az Osmocom hivatalos Git oldalán látszik az egyes repositoryk utolsó aktivitása



5-181. kép: Az Osmocom GitHub oldalán található kiemelt repositoryk

Általános adatok táblázata:

Repository neve	Első és utolsó commit között eltelt napok száma	Commitos napok száma	Leghosszabb commit nélküli időszak hossza napokban	Commitok száma összesen	Sorok száma összesen
libosmocore	3 168	1 015	46	2 553	247 129
osmo-bsc	3 591	1 627	32	6 577	772 437
osmo-bts	2 789	651	75	1 675	150 127
osmo-ggsn	5 788	197	1 877	496	344 510
osmo-msc	3 589	1 544	32	6 291	772 130
osmo-sgsn	3 584	1 488	32	6 067	707 087

Átlag értékek táblázata:

Repository neve	Átlag commit szám az összes napra vetítve	Átlag commit szám a commitos napokra vetítve	Átlag sorok száma az összes napra vetítve	Átlag sorok száma a commitos napokra vetítve	Átlag sorok száma egy commitra vetítve
libosmocore	0,805871	2,515271	78,007891	243,476847	96,799452
osmo-bsc	1,831523	4,042409	215,103592	474,761524	117,445188
osmo-bts	0,600574	2,572965	53,828254	230,609831	89,628060
osmo-ggsn	0,085695	2,517766	59,521424	1 748,781726	694,576613
osmo-msc	1,752856	4,074482	215,137921	500,084197	122,735654
osmo-sgsn	1,692801	4,077285	197,289900	475,192876	116,546399

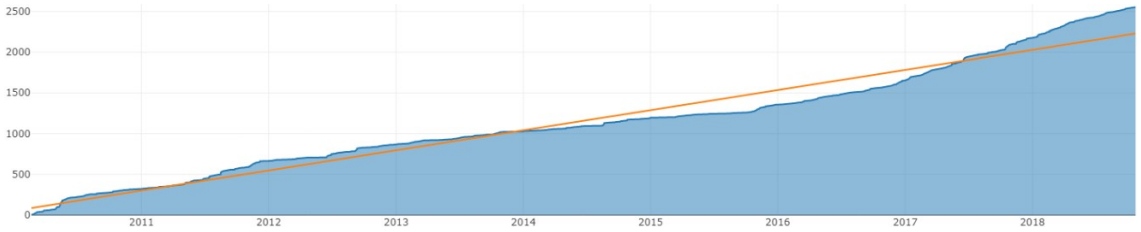
Kumulált commitok számához számított lineáris regressziós értékek:

Repository neve	Kumulált commit számok esetén vett lineáris regresszió meredeksége	Kumulált commit számok esetén vett lineáris regresszió metszéspontja	Kumulált commit számok esetén vett lineáris regresszió R-négyzet értéke
libosmocore	0,675620	88,089492	0,958979
osmo-bsc	1,541592	1 266,264919	0,931502
osmo-bts	0,579561	-77,222346	0,981458
osmo-ggsn	0,046687	44,234480	0,715750
osmo-msc	1,515179	1 291,069158	0,926485
osmo-sgsn	1,491141	1 318,569439	0,919774

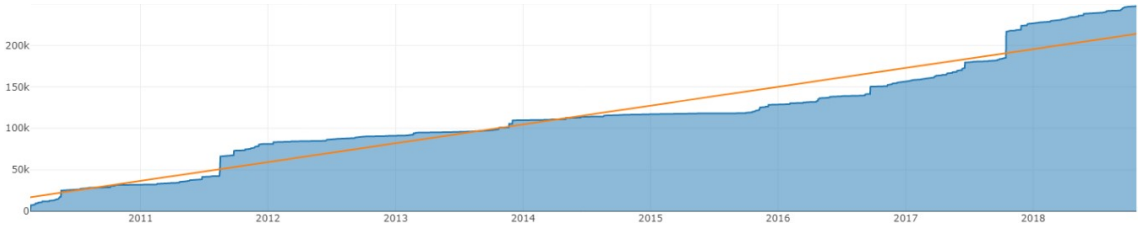
Kumulált sorok számához számított lineáris regressziós értékek:

Repository neve	Kumulált sorok száma esetén vett lineáris regresszió meredeksége	Kumulált sorok száma esetén vett lineáris regresszió metszéspontja	Kumulált sorok száma esetén vett lineáris regresszió R-négyzet értéke
libosmocore	62,120833	16 959,886493	0,918362
osmo-bsc	166,431214	106 539,575718	0,918952
osmo-bts	53,136320	-4 397,691932	0,977232
osmo-ggsn	38,185159	88 844,265394	0,902311
osmo-msc	168,409435	104 300,238444	0,915822
osmo-sgsn	161,442466	112 270,625062	0,921940

5.3.4.1 libosmocore

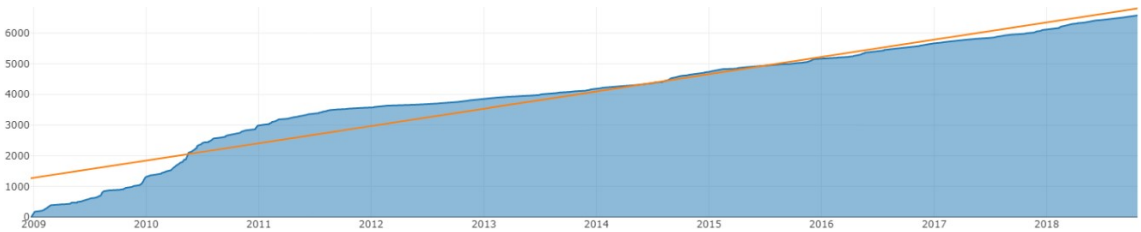


5-182. kép: libosmocore (kumulált commitok)

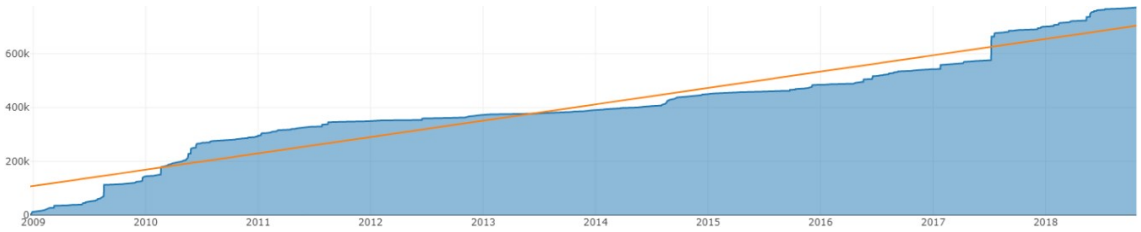


5-183. kép: libosmocore (kumulált sorok)

5.3.4.2 osmo-bsc

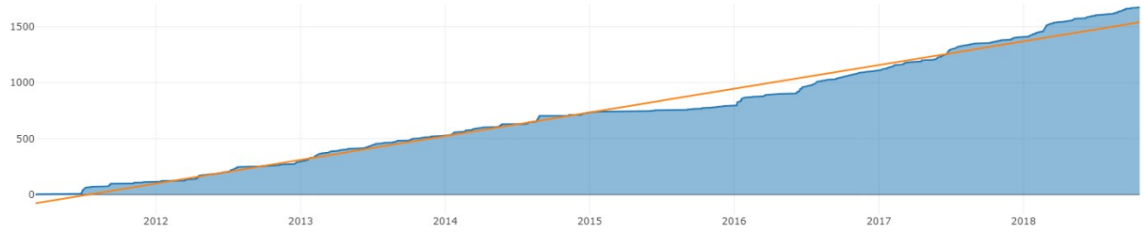


5-184. kép: osmo-bsc (kumulált commitok)

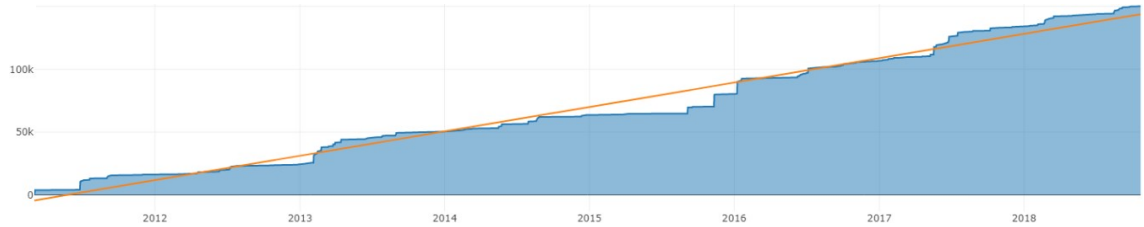


5-185. kép: osmo-bsc (kumulált sorok)

5.3.4.3 osmo-bts

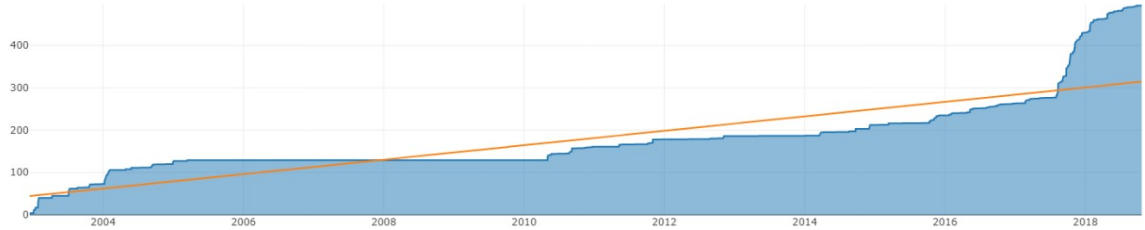


5-186. kép: osmo-bts (kumulált commitok)



5-187. kép: osmo-bts (kumulált sorok)

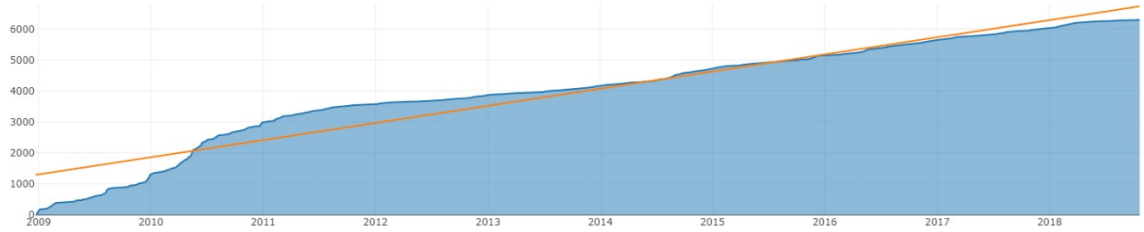
5.3.4.4 osmo-ggsn



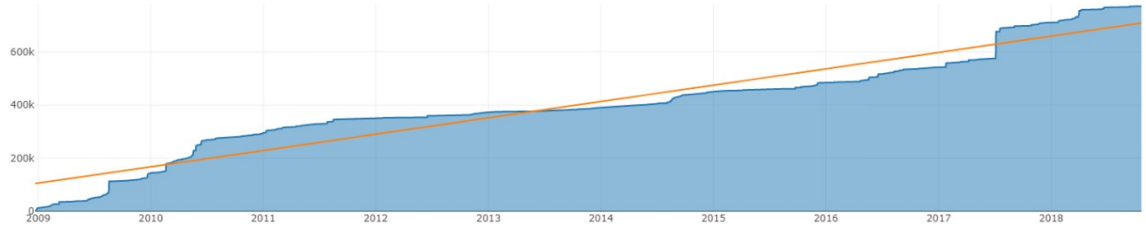
5-188. kép: osmo-ggsn (kumulált commitok)

5-189. kép: osmo-ggsn (kumulált sorok)

5.3.4.5 osmo-msc

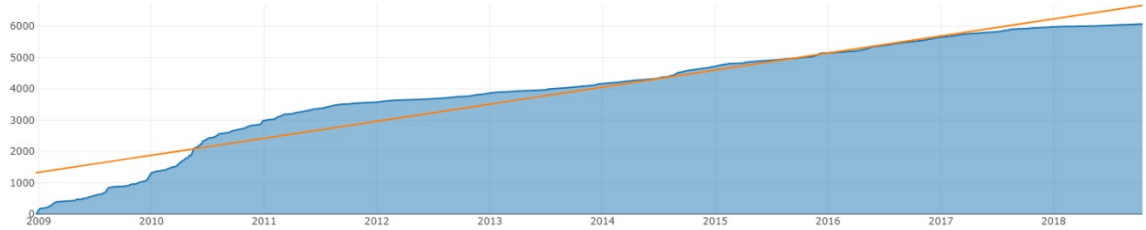


5-190. kép: osmo-msc (kumulált commitok)

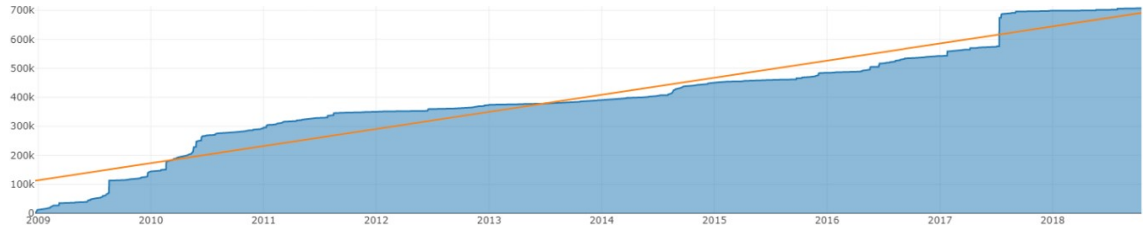


5-191. kép: osmo-msc (kumulált sorok)

5.3.4.6 osmo-sgsn



5-192. kép: osmo-sgsn (kumulált commitok)



5-193. kép: osmo-sgsn (kumulált sorok)

6 Összefoglalás

6.1 A kitűzött cél és a feladat bemutatása

Munkám célkitűzése minél több és változatosabb szoftverprojekt fejlődésének megvizsgálása volt. Ehhez olyan adatforrásra volt szükségem, amely objektíven, azonos szempontok szerint jellemzi a különböző projekteket, egyforma struktúrával írja le őket, széles projekthalmazon rendelkezésre áll és könnyen elérhető. A szoftverprojekteknél használt verziókezelő rendszerek megfelelő adatforrást jelentenek a felsorolt szempontok szerint. Munkám során az egyik legelterjedtebb verziókezelő rendszerből kigyűjtött adatok feldolgozásával vizsgáltam a különböző projektek termelékenységét és fejlődését, a forráskódban történt módosítások alapján.

6.2 A téma szakirodalmi és gyakorlati felmérése

A háttérkutatás során áttekintettem a témakör szakirodalmát, mely során a Lehman törvények mellett több olyan publikációt is találtam, amely szerint megfigyelhetőek bizonyos általános, projektekre jellemző szabályszerűségek. A projektek fejlődésével kapcsolatban kimutatható egyfajta állandóság és egyensúlyi állapot.

A szakirodalmon túl, manuális módszerekkel megvizsgáltam az ipari partner által felkínált adatforrást. Ennek eredményei, valamint a megszerzett ismeretek alapján a verziókezelő rendszerből kinyert commitok és módosított sorok számára a lineáris becslés megfelelőnek minősült. Főleg, ha figyelembe vesszük, hogy a lineáris becslés könnyen számolható és jól skálaható hosszútávon is.

6.3 A vizsgálat kiterjesztése automatizálással

A következő lépés a mérések számának növelése volt. Ehhez automatizáltam az adatok feldolgozásának folyamatát, a GitRepoAnalyzer nevű saját készítésű programmal. Ennek tervezésekor szemelőtt tartottam, hogy a létrehozott szoftver könnyen integrálható legyen további mérések és kutatások számára. Ennek érdekében egy JSON bementi és kimeneti fájlokkal dolgozó Web API alkalmazás mellett döntöttem, amely végigcsinálja a teljes mérési folyamatot, kezdve az adatforrás előkészítsével (Git repository klónozása), majd a repository előzményeinek struktúrált összegyűjtésén át egészen a különböző szempontú mérési eredmények kiszámításáig.

6.4 Mérési eredmények értékelése

Az előzetes munka során megszerzett tapasztalatot az esetek többségében megerősítették az automatizált mérések eredményei is. A vizsgált esetek többségében a commitok számának alakulása a szoftverprojekt fejlesztési időszaka alatt megközelítőleg állandó meredekséggel és lineárisan nőtt, tehát jól becsülhető lineáris megközelítéssel. A módosított sorok számán alapuló méréseket torzítják többek között a generált kódsorok, az egyszerre beérkező nagy forráskódrészletek például egy másik fejlesztési ágról, vagy az időközben verziókezelés alá vett nem forráskódot tartalmazó egyéb fájlok.

6.5 A célkitűzés áttekintése

A kitűzött cél szempontjából sikerült egy elég változatos projekthalmazt megvizsgálni. A mért projektek közt nem csak céljaikban és funkcionalításukban van sokféleség, hanem méretben, időtávban, programnyelvben és technológiában is. Sőt, nagy valószínűséggel vannak köztük különböző szoftverfejlesztési módszertanokat alkalmazó projektek, eltérő humán erőforrásokkal, jogi és anyagi háttérrel.

6.6 További kutatási lehetőségek

A bemutatott eredmények olyan további kérdéseket vetnek fel, mint például:

- A verziókezelő rendszeren keresztül mért metrikák vajon hasonlóságot mutatnak-e a termelékenység más szempontú értelmezéséhez és vizsgálatához képest?
- Mitől függ a projekt adataira legjobban illeszkedő egyenes meredeksége?
- A projekt szinten látott trendekhez hasonlókat kapnánk a fejlesztők szintjén nézve is?

Ezek megválaszolásához további kutatások szükségesek.

7 Adathordozó tartalma

A mellékelt adathordozó tartalmának kiemelt szerepű részei:

- \Diplomamunka – Zsiga Attila – 2019.pdf: A diplomamunka szövege digitális, PDF formátumban.
- \Diplomamunka – Zsiga Attila – 2019 függelék.pdf: Az összes mérési eredményt tartalmazó fájl.
- \GitRepoAnalyzer: A GitRepoAnalyzer programot tartalmazó mappa.
    - \GitRepoAnalyzer.exe: A program Windows operációs rendszerű környezetben használatos indító fájlja.
    - \appsettings.json: A program aktuális beállításait tartalmazó konfigurációs fájl.
    - \wwwroot: A webes megjelenítéshez kapcsolódó fájlokat tartalmazó mappa.
        - \index.html: A webes megjelenítés főoldala.

8 Irodalomjegyzék

[1] WHO, „WHO | Global status report on road safety 2013,” 2. december 2018.. [Online]. Available: http://www.who.int/violence_injury_prevention/road_safety_status/2013/en/. [Hozzáférés dátuma: 2. december 2018.].

[2] W. H. Organization, Global Status Report on Road Safety 2013: Supporting a Decade of Action, Luxembourg: World Health Organization, 2013.

[3] WHO, „WHO | Global status report on road safety 2015,” 2. december 2018.. [Online]. Available: http://www.who.int/violence_injury_prevention/road_safety_status/2015/en/. [Hozzáférés dátuma: 2. december 2018.].

[4] W. H. Organization, Global Status Report on Road Safety 2015, Italy: World Health Organization, 2016.

[5] D. Barnes, „The Myth of Developer Productivity — Dev9,” 2. február 2015.. [Online]. Available: https://dev9.com/blog-posts/2015/1/the-myth-of-developer-productivity. [Hozzáférés dátuma: 2. december 2018.].

[6] Y. Brikman, „The 10x developer is NOT a myth,” 29. szeptember 2013.. [Online]. Available: https://www.ybrikman.com/writing/2013/09/29/the-10x-developer-is-not-myth/. [Hozzáférés dátuma: 2. december 2018.].

[7] antirez, „The mythical 10x programmer - <antirez>,” 2. február 2017.. [Online]. Available: http://antirez.com/news/112. [Hozzáférés dátuma: 2. december 2018.].

[8] A. Oliver, „Ruby and the myth of developer productivity | JavaWorld,” 31. május 2012.. [Online]. Available: https://www.javaworld.com/article/2078578/scripting-jvm-languages/ruby-and-the-myth-of-developer-productivity.html. [Hozzáférés dátuma: 2. december 2018.].

[9] „Lehman's laws of software evolution - Wikipedia,” 6. november 2018.. [Online]. Available:

https://en.wikipedia.org/wiki/Lehman%27s_laws_of_software_evolution. [Hozzáférés dátuma: 2. december 2018.].

[10] M. M. Lehman, „Programs, Life Cycles, and Laws of Software Evolution,” *Proceedings of the IEEE,* %1. kötet68., %1. szám9., pp. 1060-1076., 1980..

[11] C. Komló, „Információs rendszerek tervezésének módszertana | Digital Textbook Library,” [Online]. Available: https://www.tankonyvtar.hu/en/tartalom/tamop412A/2011-0021_34_informacios_rendszerek_tervezesenek_modszertana/1122_a_rendszere volci_dinamikja.html. [Hozzáférés dátuma: 2. december 2018.].

[12] M. E. Conway, „Mel Conway’s Home Page,” [Online]. Available: http://www.melconway.com/Home/pdf/committees.pdf. [Hozzáférés dátuma: 2. december 2018.].

[13] M. E. Conway, „How do Committees Invent?,” *Datamation,* pp. 28-31., 1968.

[14] „Conway's law - Wikipedia,” 23. szeptember 2018.. [Online]. Available: https://en.wikipedia.org/wiki/Conway%27s_law. [Hozzáférés dátuma: 2. december 2018.].

[15] „Conway törvénye – Wikipédia,” 1. július 2018.. [Online]. Available: https://hu.wikipedia.org/wiki/Conway_t%C3%B6rv%C3%A9nye. [Hozzáférés dátuma: 2. december 2018.].

[16] E. S. Raymond, The New Hacker's Dictionary, Cambridge: MIT Press, 1991.

[17] J. O. Coplien és N. B. Harrison, Organizational Patterns of Agile Software Development, Prentice Hall, 2005..

[18] A. MacCormack, J. Rusnak és C. Baldwin, „Exploring the Duality between Product and Organizational Architectures: A Test of the “Mirroring” Hypothesis,” *Research Policy,* %1. kötet41., %1. szám8., pp. 1309-1324., 2012..

[19] N. Nagappan, B. Murphy és V. R. Basili, „The Influence of Organizational Structure on Software,” január 2008.. [Online]. Available:

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2008-11.pdf. [Hozzáférés dátuma: 2. december. 2018.].

[20] A. Kovács és K. Szabados, „Internal quality evolution of a large test," *Acta Universitatis Sapientiae, Informatica,* %1. kötet8., %1. szám2., pp. 216-240, 2016.

[21] G. Farkas, „Amit tudnod kell fejlesztőként, IV. rész: Verziókezelés | ITHub," 11. november 2014.. [Online]. Available: https://ithub.hu/blog/post/Amit_tudnod_kell_fejlesztokent_IV_resz_Verziokezeles/. [Hozzáférés dátuma: 2. december 2018.].

[22] R. Potvin és J. Levenberg, „Why Google Stores Billions of Lines of Code in a Single Repository," *Communications of the ACM,* %1. kötet59., %1. szám7., pp. 78-87., 2016..

[23] R. Potvin és J. Levenberg, „The ACM Digital Library," július 2016.. [Online]. Available: http://delivery.acm.org/10.1145/2860000/2854146/p78-potvin.pdf?ip=84.236.37.101&id=2854146&acc=OA&key=4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E5945DC2EABF3343C&__acm__=1543768516_889efad909bdf11bd4d2786c7f13fe78. [Hozzáférés dátuma: 2. december 2018.].

[24] J. Kabanov, „Developer Productivity Report 2013 zeroturnaround.com," 18. szeptember 2013.. [Online]. Available: https://zeroturnaround.com/rebellabs/developer-productivity-report-2013-how-engineering-tools-practices-impact-software-quality-delivery/10/. [Hozzáférés dátuma: 2. december 2018.].

[25] MentorMate, „SVN, Git Out of Here: How to Choose Your Version Control System," 8. február 2017.. [Online]. Available: https://medium.com/@MentorMate/svn-git-out-of-here-how-to-choose-your-version-control-system-10b44750a75c. [Hozzáférés dátuma: 2. december 2018.].

[26] RhodeCode, „RhodeCode › Version Control Systems Popularity in 2016," 2016.. [Online]. Available: https://rhodecode.com/insights/version-control-systems-2016. [Hozzáférés dátuma: 2. december 2018.].

[27] B. Thompson, „Git didn't beat SVN, GitHub did. | GitPrime Blog,” [Online]. Available: https://blog.gitprime.com/git-didnt-beat-svn-github-did/. [Hozzáférés dátuma: 2. december 2018.].

[28] Google, „Git, SVN, Subversion, Mercurial, TFVC - Felfedezés - Google Trends,” [Online]. Available: https://trends.google.com/trends/explore?date=all&q=Git,SVN,Subversion,Mercurial,TFVC. [Hozzáférés dátuma: 2. december 2018.].

[29] A. Kovács, „Testing at Scale,” 8. december 2018.. [Online]. Available: http://compalg.inf.elte.hu/~attila/TestingAtScale.htm. [Hozzáférés dátuma: 8. december 2018.].

[30] Osmocom, „Open Source Mobile Communications,” 2006-2018.. [Online]. Available: http://osmocom.org/. [Hozzáférés dátuma: 2. december 2018.].

[31] Osmocom, „Official Osmocom mirror,” 2018.. [Online]. Available: https://github.com/osmocom. [Hozzáférés dátuma: 2. december 2018.].

[32] Osmocom, „osmocom.org git repositories,” 2. december 2018.. [Online]. Available: http://git.osmocom.org/. [Hozzáférés dátuma: 2. december 2018.].