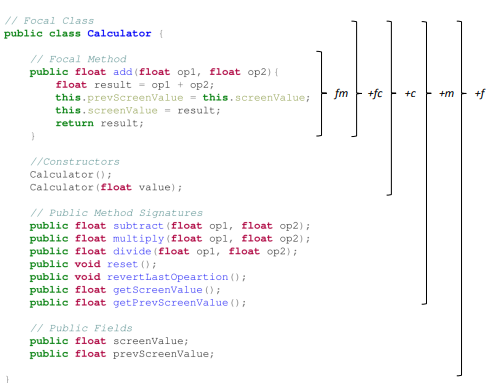
Дипломна работа

# Анализ на съществуващи решения

## AthenaTest [https://arxiv.org/pdf/2009.05617.pdf%20%E2%80%93%20unit%20Test%20Case%20Generation%20with%20Transformers%20and%20Focal%20Context]

Генерация на модулно тестване използвайки трансформатори и фокусен контекст е разработка на Мишел Туфано (Michele Tufano), Даун Дрейн (Dawn Drain), Алексей Святковски (Alexey Svyatkovski), Шао Кун Денг (Shao Kun Deng) и Нийл Съндерсън (Neel Sundersan).

Тази разработка се различава от други с това, че не е базирана на метрика за покритие на тестовете, а на контекста, който трябва да бъде валидиран от тест. При тази разработка подходът е от тип “sequence to sequence”, който се състои от процес с две стъпки при тренирането на модела. Първата стъпка е тренирането на модел върху голям обем от Java код, който после да бъде настроен фино (fine-tuning) за задачата на генериране на код. За целта е и създаден най-големият набор от данни, който е ѝ именуван, за модулни тестове с техните контексти – тестван метод, дефиниция на класа на тествания метод, конструктор на тествания клас, методи на тествания клас и полета. Данните са кръстени Methods2Test и могат да бъдат намерени и използвани свободно в GitHub.



Целият процес за генериране на тестове е както следва:

1. Събират се данни – съществуващи тестове и контексти,
2. Тестовете се асоциират с техните контексти
3. Използва се общият модел
4. Създава се фината настройка, след което вече се генерират тестовете

Процеса съществува на фигурата отдолу:

Graphical user interface, diagram

Description automatically generated

## Github Copilot [https://copilot.github.com]

Github Copilot е приложение, чиято цел е да бъде пълноценен помощник, който да се държи като втори програмист, който програмира с потребителя. Самото приложение е имплементирано като притурка (plugin) на средата за разработка Visual Studio Code. GitHub Copilot работи по механизъм подобен на този за обикновеното допълване при въвеждане на код, само че вместо да допълва една дума може да допълни цяла функция или метод на база на дадено описателно име и/или коментар, който описва действието на съответния фрагмент от код.

Реално GitHub Copilot представлява генератор на код, който генерира код от дадено описание. Това е постигнато с помощта на изкуствен интелект и по-конкретно с модел разработен от OpenAI в колаборация с Microsoft. Моделът се казва Codex и е базиран на огромния модел с 12 милиарда параметъра трансформатор (transformer) GPT-3. Моделът е трениран използвайки хиляди публични хранилища за код включително и всички такива от GitHub. Моделът, съответно и GitHub Copilot, се справя най-добре с код на Python, тъй като най-много такива данни са присъствали при тренирането на модела.

За да може да генерира предложения GitHub Copilot използва отворения файл в Visual Studio Code, както и другите файлове, които са част от отвореното работно пространство. С този контекст моделът може да направи предложение, което е конкретно за контекста на проекта.

На практика продуктът работи най-добре при писане на малки функции с описателни имена и точно описание на какво трябва да извърши функцията.



## EvoSuite [https://www.evosuite.org/evosuite/]

EvoSuite е система, която може автоматично да генерира модулни тестове за класове написани на Java. Това се постига, чрез използване на хибриден подход, който генерира и оптимизира цели тестови пакети, които се стремят към постигане на високо покритие върху тествания клас. Самите тестове са малки по размер и ефективни използвайки достатъчно клаузи от тип “assert”, които да са достатъчни за описанието и поддръжката на съществуващата логика. Системата има за цел да предоставя тестове, на които разработчиците да разчитат при по нататъшни пренаписвания на различни фрагменти. Тези тестове също се очаква да гарантират сегашната работа на софтуера.

Системата има следните функционалности:

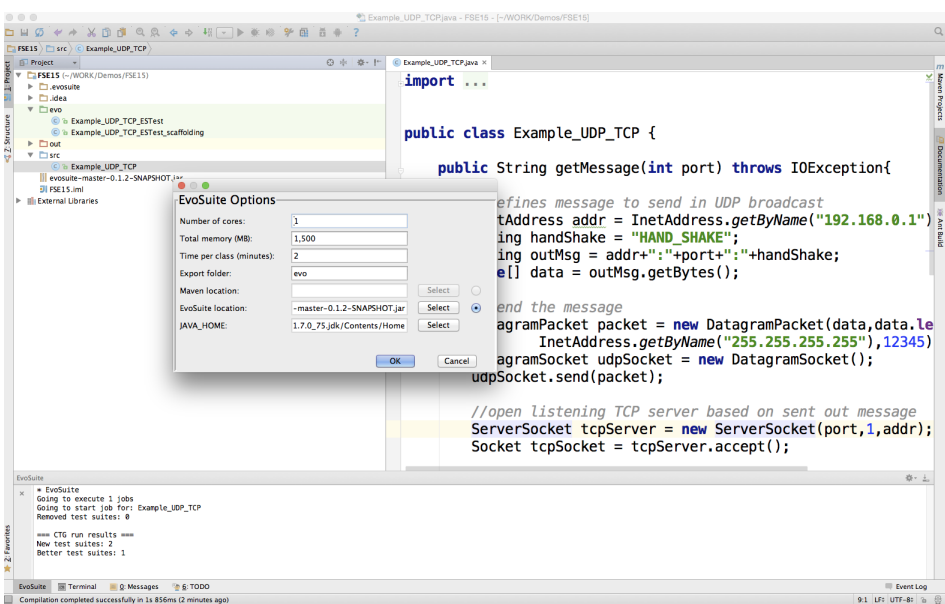
* Генериране на Junit4 тестове за избрани класове
* Оптимизация на тестове за конкретна критерия на покритие – линии, разклонения, изходи и тестове за мутации
* Генерираните тестове се минимизират – размерът им се свежда до минимум запазвайки същото ниво на покритие
* Генериране на Junit проверяващи клаузи, които да проверяват моментното състояние на класа
* Пускане на тестовете във виртуална среда за по-голяма сигурност
* Виртуална файлова система
* Виртуални компютърна мрежа

Системата може е достъпна по няколко начина:

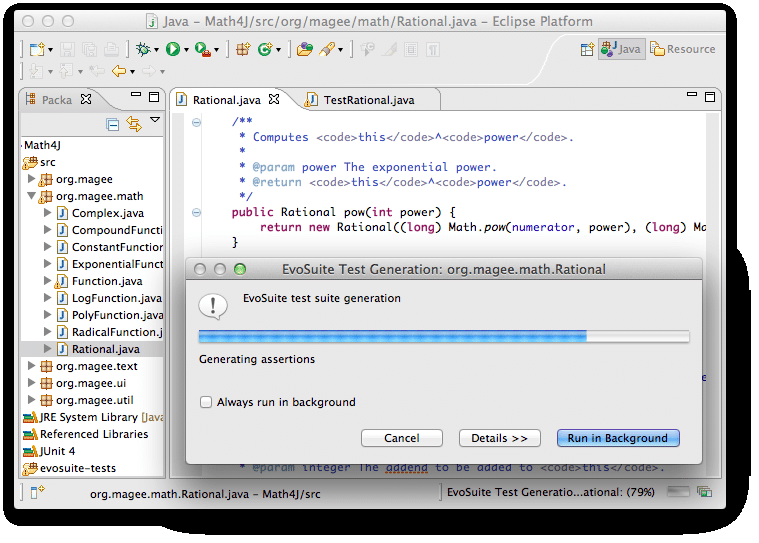
Първият, от които е като инструмент с конзолен интерфейс, за да се използва по този начин се сваля .jar файл, чрез който се използва системата.

Вторият вариант е като притурка (plugin) към средата за разработка. Към момента такива притурки съществуват за InteliJ IDEA и Eclipse.

След като се инсталира нужната притурка за InteliJ IDEA, тестове се генерират като се избере клас, група от класове или пакет и чрез десен клик на мишката се избере генериране на EvoSuite Test. Потребителският интерфейс изглежда по следния начин:



При Eclipse приложението работи по аналогичен начин:



Освен тези два плъгина съществува такъв и за системата за непрекъсната интеграция – Jenkins. С нея може да се добави стъпка, която да генерира и изпълнява тестове за всички новодобавени класове.

## Randoop[https://randoop.github.io/randoop/]

Randoop е система за генериране на модулни тестове за Java. Системата генерира тестове с Junit. Част от функционалността на Randoop съществува и за .Net.

Алгоритъмът, който използва Randoop за генериране на тестове е чрез итеративно генерация на произволни тестове, като се използва обратна връзка от генерирания тест, за да се подобри и стигне до финалната му форма. На практика се генерират псевдо-произволни извиквания към методите и конструкторите на тествания клас, след което генерираният тест се пуска и спрямо резултатите от извиканите методи се създават проверки.

Тази система се използва главно за намиране на бъгове в съществуващите системи. Често пъти генерираните тестове не са лесно четими за програмистите и за това се използват основно за намиране на регресии и бъгове.

## Evaluating Large Language Models Trained on Code – Codex

# Функционално описание на разработката

## Цел

Целта на настоящата дипломна работа е разработката на система, която да генерира тестове базирани на критериите описани в потребителската история, която е разработвана от програмиста. Генерираните тестове са функционални, интеграционни и в някои случаи на цялата система – end to end. Идеята да съществува такъв инструмент е да се подобри продуктивността на програмистите в ежедневната им работа като се намали времето, което те инвестират в писане на автоматични тестове, а освен това и да се подобри качеството на разработвания продукт.

Идеята за автоматична генерация на код не е нова, както е описано в първа глава на дипломната работа съществуват разработки с различни подходи на тази тематика. От използване на изкуствен интелект до системи, които генерират псевдо произволни извиквания на съществуващи методи.

Важно е да се отбележи, че разработките, които съществуват на тематиката за генерация на тестове се фокусират основно и само върху генерация на модулни тестове. При тези проекти най-важно е разбирането за модула (класа) и няма нужда от знание за цялата системата докато в системата, която се разработва за тази дипломна работа е важно да се тества цяла функционалност, а не конкретен модул.

Решението, което е разработено в тази дипломна работа е добра демонстрация на концепцията – използване на дълбоко обучение за генериране на тестове. Към момента то може да се използва като начална версия на тестове, които да бъдат доразработени от програмиста или съответно инженерите по качеството.

За програмната реализация на проблема с генериране на код за функционални тестове са използвани редица технологии. Основните технологии използвани за продукта са Python и модели разработени от OpenAI, както и популярният стек за работа с големи обеми от данни за Python, а именно Numpy, Pandas и популярната библиотека за работа с естествени езици NLTK. Освен тези библиотеки са използвани и синтактични дървета както и библиотеката PyDriller за по-лесна обработка на хранилища за код от тип git.

## Език за програмиране

Езикът за програмиране, който е избран за реализацията на дипломната работа е python. За разработката на този софтуер това е един от най-добрите избори, защото е най-популярният сред обществото, което се занимава с изкуствен интелект и анализ на данни. Нормалното следствие от популярността на езика в тези среди е и изключително добрата му поддръжка на тематиката изразяваща се в множество различни библиотеки, които се разработват за анализ и обработка на данни както и такива свързани с машинно обучение.

## Библиотеки

### NLTK

Библиотеката, която е избрана за ОЕЕ е NLTK. NLTK е една от водещите платформи за разработка на програми на Python, които работят с данни от естествен език. Библиотеката предоставя лесни за използване интерфейси включвайки в себе си множество ресурси от данни. Библиотеката съдържа в себе си множество под-библиотеки за класификация, токенизация, коренуване, парсиране, определяне на частите на речта, определяне на „емоцията “ и други. Основните причини за избора на тази библиотека е това, че тя е създадена за Python и в себе си съдържа всичко, което би потрябвало в разработката на дипломната работа и далеч повече освен това е една от най-използваните, с което идват и множеството ресурси свързани с нея.[17]

### Pandas [https://www.nobledesktop.com/learn/python/pandas-overview, https://pandas.pydata.org/about/]



Pandas е библиотека с отворен код, чието начало е поставено през 2008 година като през 2009 кодът на библиотеката става публично достъпен. Името на библиотеката идва от думите на английски за панел и данни - “**Pan**el” и “**Da**ta”. Библиотеката предоставя функционалности за обработка на големи масиви от данни. Основни нейни функционалности са:

* Бързи и ресурс ефективни обекти от тип рамки за данни (DataFrame), които са предвидени за манипулация и индексация
* Инструменти за сериализация и десериализация – запазване на данните в често използвани формати като CSV, Microsoft Excel, SQL и HDF5
* Автоматична подредба на данните като се използват етикети, а не само цифрови индекси. Библиотеката следи за липсващи данни.
* Размера на матрицата с данни може да бъде сменян лесно и бързо. Т.е. операции с модификация на колони и редове са позволени и работят бързо.
* Мощен механизъм за групиране (филтриране) на данните по критерии
* Висока производителност при сливане на големи обеми от данни
* Висока производителност на библиотеката – имплементирана е с технологии от ниско ниво, които да подсигурят тази производителност, а именно C и Cython.
* Една от най-използваните библиотеки в академичните среди. Широко използвана в много индустрии – Финанси, Икономика, Невронаука, Статистика, Реклама, Анализ на големи данни.

Тази библиотека бе избрана за разработката на дипломната работа тъй като реализацията на генериране на автоматични тестове изисква работа с голям обем от данни – съществуващ и генериран код.

### PyDriller [https://pydriller.readthedocs.io/en/latest/]

PyDriller е библиотека, която дава абстракция и интерфейс върху командите и работата с Git хранилище. Идеята на библиотеката е да предостави възможност на различните разработчици да анализират и обработват данни от различни хранилища на код по лесен начин.

Библиотеката се състои от 3 основни обекта – хранилище (Repository), ревизия/версия (Commit) и модифициран файл (Modified File).

За целта на дипломната работа се използват различните ревизии и съответно от всяка се разграничават по-съществените файлове, тези които са медиатори и дефинират конкретна функционалност от тези, които са листа на дървото на зависимостите на софтуера.

### AST

## Open AI

# Описание на реализацията на разработката

# Използвани данни и проведени експерименти

# Примерна употреба и валидация на системата