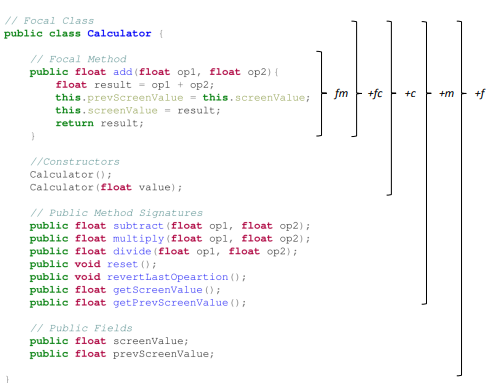
Дипломна работа

# 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:

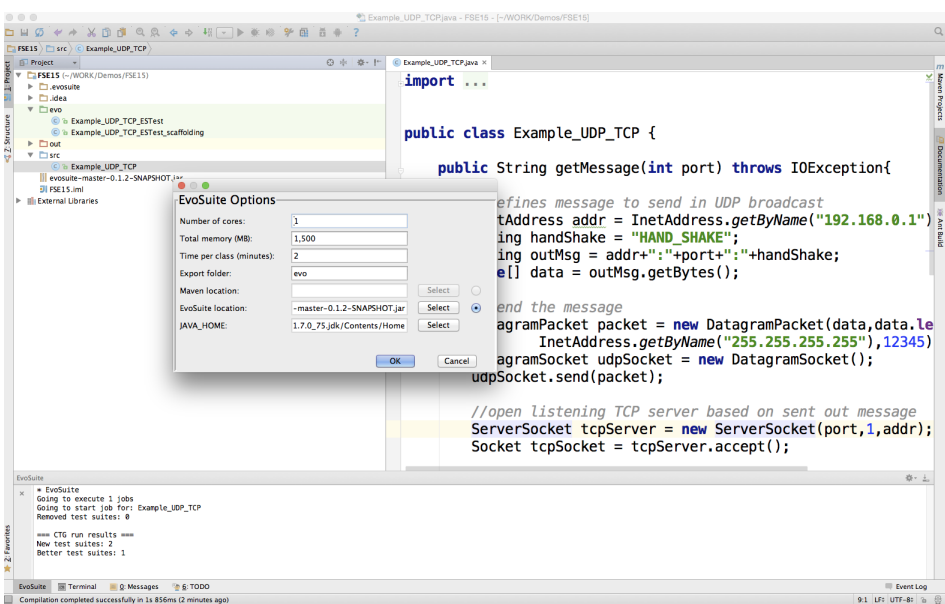
* Генериране на Junit4 тестове за избрани класове
* Оптимизация на тестове за конкретна критерия на покритие – линии, разклонения, изходи и тестове за мутации
* Генерираните тестове се минимизират – размерът им се свежда до минимум запазвайки същото ниво на покритие
* Генериране на Junit проверяващи 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:

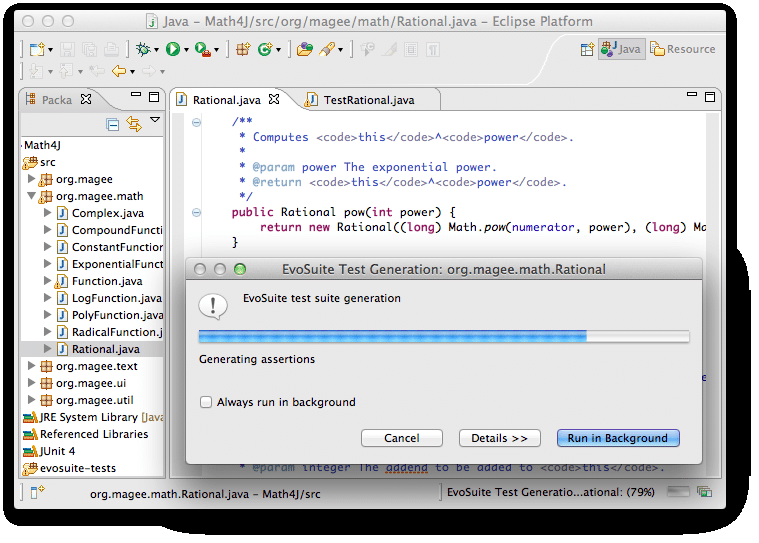
Първият, от които е като инструмент с конзолен интерфейс, за да се използва по този начин се сваля .jar файл, чрез който се използва системата.

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.

# Функционално описание на разработката

# Описание на реализацията на разработката

# Използвани данни и проведени експерименти

# Примерна употреба и валидация на системата