ZERO-SHOT UNIT TEST GENERATION WITH LARGE LANGUAGE MODELS

IVAYLA Y. PENCHEVA

ABSTRACT: *This paper investigates zero-shot unit test generation using three large language models (LLMs): ChatGPT (GPT-5.3), GitHub Copilot Pro, and Google Gemini 3.1 Pro. The study evaluates 27 real-world defects from four projects in the Defects4J benchmark (Lang, Math, Time, and Chart), generating a total of 81 test cases without any information about the nature of the defects. The results show that all three models achieve similar defect detection rates (22–26%), suggesting that model choice has limited impact under zero-shot conditions. Two categories of defects are identified: transparent defects, which can be detected without additional guidance, and hidden defects, whose detection requires specific contextual information and for which the detection rate is low. Zero-shot data may provide a basis for future ML models that predict defect detectability.*

KEYWORDS: *large language models, unit testing, zero-shot prompting, defect detection, Defects4J, software testing*

DOI: https://doi.org/10.46687/ENQH1947

ZERO-SHOT ГЕНЕРИРАНЕ НА UNIT ТЕСТОВЕ С ГОЛЕМИ ЕЗИКОВИ МОДЕЛИ*

ИВАЙЛА Й. ПЕНЧЕВА

АБСТРАКТ: *Настоящата статия изследва генерирането на unit тестове в zero-shot режим с три големи езикови модела: ChatGPT (GPT-5.3), GitHub Copilot Pro и Google Gemini 3.1 Pro. Анализирани са 27 реални дефекта от четири проекта в Defects4J, като са генерирани 81 тестови случая без информация за естеството на дефектите. Резултатите показват, че и трите модела постигат сходна успеваемост при откриване на дефекти (22–26%), което предполага, че изборът на модел има ограничено влияние при zero-shot условия. Идентифицирани са две категории дефекти: прозрачни, които могат да бъдат открити без допълнителни насоки, и скрити, чието откриване изисква специфична контекстна информация и при които успеваемостта е ниска. Данните от zero-shot режима отразяват реалистични условия и могат да служат като основа за ML модели, предсказващи откриваемостта на дефекти.*

КЛЮЧОВИ ДУМИ: *големи езикови модели, unit тестване, промпт без примери (zero-shot), откриване на дефекти, Defects4J, софтуерно тестване*

1. Въведение

Автоматичното генериране на тестове с помощта на големи езикови модели (LLM) се превръща в активно изследвана област в съвременното софтуерно инженерство. Модели като ChatGPT, GitHub Copilot и Google Gemini демонстрират способност да генерират unit тестове за непознат код, което отваря възможности за значително ускоряване на процеса на тестване. Unit тест (модулен тест) проверява отделен метод или клас изолирано от останалата част на системата.

Въпреки нарастващия интерес, повечето съществуващи изследвания оценяват AI моделите при условия с подробна информация - описание на дефекта, на задействащия вход

* Настоящата статия е частично финансирана по проект № РД-08-91/02.02.2026 г. на фонд Научни изследвания на ШУ „Епископ Константин Преславски“

или на очакваното поведение [2, 3]. В реалната практика обаче разработчикът рядко разполага с тази информация предварително. Остава открит въпросът: какво могат AI моделите да постигнат без никаква информация за дефекта?

Настоящата статия отговаря на този въпрос чрез систематично емпирично изследване върху 27 реални дефекта от Defects4J [1]. За всеки дефект три AI модела генерират по един тест, без да получават информация за дефекта (режим zero-shot), като резултатите се оценяват по строг бинарен индикатор P - дали тестът открива дефекта при дефектната версия и преминава при поправената.

Статията има три приноса - систематична оценка на три AI модела върху 81 теста, генерирани без информация за дефекта, класификация на дефектите на прозрачни и скрити според това дали се откриват без такава информация и емпирично потвърждение, че покритието на кода не корелира с откриването на дефекти. Отделно, резултатите подсказват, че данните от Ниво 0 отразяват по-реалистични условия и затова са по-подходяща основа за обучение на ML модели.

2. Свързани изследвания

2.1. Defects4J

Defects4J [1] е публична колекция от реални дефекти в Java програми, за всеки от които са налични дефектна и поправена версия заедно със съответната тестова инфраструктура. Съставът ѝ е фиксиран, което позволява различни изследвания да се сравняват върху едни и същи дефекти - затова колекцията се използва като еталон (benchmark) за техники за автоматично генериране и приоритизиране на тестове.

2.2. Генериране на тестове с LLM

Съществуващите изследвания в областта на автоматичното генериране на тестове с LLM се фокусират предимно върху условия с подробна информация за дефекта. ChatUniTest [2] използва ChatGPT за генериране на unit тестове с контекстна информация за метода и неговите зависимости. Cover-Agent [3] комбинира LLM с анализ на покритието за итеративно подобряване на тестовите пакети. AthenaTest [4] изследва способността на LLM да генерират тестове на базата на описание на фокусния метод.

За разлика от тези подходи, настоящото изследване се фокусира върху zero-shot условия - без информация за конкретния дефект, неговото местоположение или начина, по който се проявява.

2.3. Приоритизиране на тестове

Приоритизирането на тестови случаи (Test Case Prioritization, TCP) е активно изследвана област с цел ранно откриване на дефекти [5]. Традиционните подходи разчитат на покритие на кода, история на изпълнението или честота на промените. Настоящото изследване допринася към тази област, като идентифицира характеристиките на дефектите, които влияят върху успеваемостта на AI-генерираните тестове.

3. Методология

3.1. Дефекти и проекти

В изследването са включени 27 реални дефекта от четири проекта в Defects4J [1]: Apache Commons Lang (Lang) - 5 дефекта, Apache Commons Math (Math) - 7 дефекта, Joda-Time (Time) - 8 дефекта и JFreeChart (Chart) - 7 дефекта. Избраните дефекти обхващат седем

типа: грешки в конструктори, логически/гранични условия, числова точност, състояние/кеширане, обработка на часови зони, объркване на типове и грешки в API/генерични типове. Това разнообразие позволява анализът да обхване различни характеристики на дефектите.

Таблица 1
Дефекти по проект и тип

Проект	Брой дефекти	Типове дефекти
Lang	5	Логически/гранични (2), конструктори (1), объркване на типове (1), API/генерични типове (1)
Math	7	Логически/гранични (3), числова точност (2), състояние/кеширане (1), объркване на типове (1)
Time	8	Часови зони (6), състояние/кеширане (1), числова точност (1)
Chart	7	Логически/гранични (3), състояние/кеширане (2), конструктори (1), часови зони (1)
Общо	**27**	**7 типа**

3.2. AI модели

Изследвани са три големи езикови модела, представляващи актуалното състояние на AI-асистираната разработка на софтуер: ChatGPT (GPT-5.3), GitHub Copilot Pro и Google Gemini 3.1 Pro. Всички модели са използвани чрез уеб интерфейсите им през март 2026 г. За всеки дефект всеки модел генерира по един тестов клас на Ниво 0 (zero-shot), което дава общо 81 задачи за генериране на тестове (27 дефекта × 3 модела). Този подход симулира реалното използване от разработчик, който генерира един тест за непознат метод.

3.3. Ниво на входната заявка към модела (промпт)

Промптовете в по-широкото изследване са разделени на нива според количеството информация за дефекта, която предоставят. Настоящата статия се фокусира изключително върху заявки от Ниво 0 - zero-shot промптове, които предоставят само сигнатурата на метода и кратко описание на очакваното му поведение, без никаква информация за естеството или местоположението на дефекта. Това отразява реалистични условия, при които разработчикът иска от AI асистент да генерира тестове за непознат метод, без предварителни познания за съществуващи грешки.

Пример за промпт от Ниво 0 за метода DateTimeFormatter.parseInto() (дефект Time-16):

> *"Generate a JUnit 4 test class named TestDateTimeFormatter in package org.joda.time.format for the DateTimeFormatter.parseInto() method in Joda-Time. Write tests that: (1) create a MutableDateTime base instant, (2) use a DateTimeFormatter with various patterns, (3) call parseInto() and assert the result is correct."*

3.4. Процедура

За всяка от 81-те комбинации (27 дефекта × 3 модела) е следвана шестстъпкова процедура. Анализът на тестовете се извършва по три условия: (У1) дали входните стойности водят до изпълнение на дефектния участък от кода, (У2) дали при дефектната версия се наблюдава грешен резултат и (У3) дали assertion-ът, т.е. твърдението, с което тестът сравнява получения и очаквания резултат, улавя грешния резултат при дефектната версия. Останалите стъпки включват избор на дефект чрез извличане на дефектната и поправената версия от Defects4J, генериране на тест чрез промпт от Ниво 0, изпълнение на

теста, измерване на индикатора P, на покритието на кода C, както и документиране на причините за неуспех при тестове с P=0.

С цел осигуряване на консистентност, условията У1–У3 са оценявани по фиксиран и унифициран протокол за всички 81 теста. Всички експерименти са проведени в детерминистична среда с фиксирани версии на Defects4J.

3.5. Дефиниция на индикатора за откриване на дефекта

Индикаторът за откриване на дефекта, означаван с P, се дефинира като бинарна променлива:

- **P = 1** - тестът не преминава при дефектната версия И преминава при поправената версия;
- **P = 0** - всички останали случаи: не преминава при двете версии, преминава при двете, не се компилира или не е приложим.

3.6. Класификация на причините за неуспех

За всички тестове с P=0 е идентифицирана причината за неуспех:

Таблица 2
Причини за неуспех

Причина	Описание
Липсващ релевантен вход	Тестът не упражнява кодовия път, съдържащ дефекта
Слаб assertion	Тестът изпълнява дефекта, но не проверява засегнатия изход
Грешен тип вход	Типът или стойността на входа не задействат дефекта
Некомпилируем код	Генерираният тест съдържа синтактични или API грешки

4. Резултати

4.1. Обща успеваемост при откриване на дефекти

Таблица 3 представя процента на тестовете, които успешно откриват дефекта (P-rate) за всеки проект и модел. Общо 81 теста са анализирани - по един тест от всеки модел за всеки от 27-те дефекта.

Таблица 3
Дял на тестовете, открили дефекта (P-rate), по проект и модел (Ниво 0, без информация за дефекта)

Проект	ChatGPT	Copilot	Gemini	Средно
Lang (5 дефекта)	0%	40%	0%	13%
Math (7 дефекта)	29%	14%	43%	29%
Time (8 дефекта)	25%	12%	12%	17%
Chart (7 дефекта)	29%	43%	43%	38%
Общо (27 дефекта)	**22%**	**26%**	**26%**	**25%**

Ключово наблюдение: разликата между трите модела е само 4 процентни пункта (от 22% за ChatGPT до 26% за Copilot и Gemini). Тези резултати не показват ясно преимущество на един от изследваните модели при разглежданите zaro-shot условия, тоест - без информация за дефекта изборът на конкретен AI модел няма съществено влияние върху успеваемостта му.

4.2. Успеваемост по тип дефект

Таблица 4 показва P-rate по тип дефект за всички модели. Резултатите разкриват ясно разделение между прозрачни и скрити дефекти.

Таблица 4
Дял на тестовете, открили дефекта (P-rate), по тип дефект

Тип дефект	Открити/Общо	P-rate	Категория
Конструктори	4/6	67%	Прозрачен
Логически/гранични	9/24	38%	Прозрачен
API/генерични типове	1/3	33%	Прозрачен
Състояние/кеширане	3/12	25%	Граничен
Числова точност	2/9	22%	Граничен
Часови зони	1/21	5%	Скрит
Объркване на типове	0/6	0%	Скрит

Дефектите от тип „Конструктори“ се откриват в 67% от случаите - те съдържат грешки от копиране или очевидно неправилни стойности, които водят до генериране на тестове, откриващи дефекта дори без такава информация. За разлика от тях, дефектите от тип „Объркване на типове“ не се откриват от нито един модел (0%), а дефектите от тип „Часови зони“ - само в 5% от случаите.

4.3. Разпределение на дефектите според броя на моделите, които ги откриват

От 27-те дефекта, само два са открити от всички три модела: Chart-20 и Math-3. Chart-20 е дефект в конструктор, а Math-3 - логически/граничен; и при двата грешната стойност е очевидна.

В другата крайност, 15 дефекта не са открити от нито един модел, сред които двата дефекта от тип „Объркване на типове“ (Lang-3, Math-75) и шест от седемте дефекта в часови зони (Time-7, Time-19, Time-20, Time-23, Time-24 и Chart-8).

4.4. Връзка между покритието на кода и откриването на дефекти

Анализът потвърждава липсата на корелация между покритието на кода (C%) и индикатора P. Показателен пример: Lang-1 постига C=98.1% при P=0 за всички модели, докато Lang-4 постига P=1 за Copilot при C=100%. Идентично покритие може да съответства на напълно различна успеваемост при откриване на дефекти.

5. Анализ на резултатите

5.1. Причини за сходната успеваемост на моделите

Резултатите показват, че без информация за дефекта всички три модела разчитат на едни и същи повърхностни признаци - имена на методи, типове параметри и стандартни практики за тестване. При тези условия трудните дефекти (часови зони, объркване на типове) остават неоткрити за всички модели, а по-лесните (конструктори, логически/гранични) се откриват по-често. Това изравнява резултатите и обяснява минималната разлика от 4 процентни пункта между моделите.

5.2. Сравнение със съществуващи изследвания

Съществуващи изследвания като ChatUniTest и Cover-Agent съобщават за значително по-високи нива на откриване на дефекти при условия с насочващи промптове (prompt-guided conditions) [2,3]. За разлика от тях, резултатите от настоящото изследване показват, че при строго zero-shot условия производителността на LLM е значително по-ниска (22–26%) и до голяма степен сходна между отделните модели. Това предполага, че докладваните по-високи резултати могат да зависят в значителна степен от качеството на промпта и от предоставената контекстна информация, а не единствено от присъщите възможности на моделите. Възможно обяснение е, че повечето съществуващи подходи използват допълнителен контекст или насочващи детайли в промпта, докато настоящото изследване разглежда строго zero-shot сценарий. Получените резултати частично противоречат на изследвания, които съобщават за висока ефективност на LLM-базираното генериране на тестове, което подчертава значението на експерименталните условия при интерпретирането на резултатите.

5.3. Прозрачни и скрити дефекти

Категоризацията на дефектите е емпирична и се основава на наблюдаваните закономерности в откриваемостта при трите изследвани модела.

Прозрачни дефекти (конструктори, логически/гранични, API/генерични типове) се характеризират с независимост от външни зависимости - грешката е видима директно от сигнатурата на метода или от стандартните гранични стойности. AI моделите ги откриват, защото генерират стандартни тестове, които случайно покриват дефектния случай.

Скрити дефекти (часови зони, объркване на типове) изискват специфична контекстна информация - конкретна часова зона, точен числов тип или специфична последователност от извиквания. Без тези допълнителни данни нито един модел не достига до съответния случай, което обяснява P-rate от 0–5%.

5.4. Практически следствия

Резултатите имат два важни практически извода. Първият е, че при автоматично генериране на тестове без информация за дефекта изборът на AI модел има ограничено значение - по-важен е типа на дефекта, който се търси. А вторият е, че за сложни дефекти (часови зони, объркване на типове) е задължително предоставянето на допълнителна контекстна информация в самия промпт.

5.5. Следствия за приложението на машинно обучение

От методологична гледна точка, резултатите потвърждават, че данните от Ниво 0 (без информация за дефекта) са по-подходяща основа за обучение на ML модели за предсказване на P. Използването на данни от Ниво 2 (с пълна информация за дефекта) би довело до изкуствено завишени стойности на P, които не отразяват реалните условия на тестване.

5.6. Заплахи за валидността

Таблица 5
Заплахи за валидността

Тип	Заплаха	Смекчаване
Вътрешна	Верификация на поправената версия	Следвана стриктна процедура от 6 стъпки
Външна	Само 4 проекта от Defects4J	Разнообразие от 27 дефекта и 7 типа
Външна	Само 3 AI модела, март 2026	Избрани най-популярните модели

Конструктивна	Дефиниция на индикатора P	Стриктна бинарна дефиниция (FAIL buggy + PASS fixed)

6. Заключение

Настоящото изследване анализира способността на три големи езикови модела да генерират unit тестове в zero-shot условия върху 27 реални дефекта от Defects4J. Резултатите показват сходна успеваемост между моделите, което предполага, че при липса на допълнителна контекстна информация характерът на дефекта оказва по-силно влияние върху откриваемостта му, отколкото изборът на конкретен модел. Наблюдава се ясно разграничение между дефекти, които могат да бъдат открити чрез стандартни тестови шаблони, и такива, които изискват специфичен контекст или домейн-специфични знания. Анализът също така потвърждава липсата на пряка зависимост между покритието на кода и успешното откриване на дефекти.

Получените резултати могат да послужат като основа за бъдещи изследвания върху предсказване и приоритизиране на индикатора P на тестовете, генерирани от изкуствен интелект, тъй като данните от Ниво 0 представляват по-подходяща тренировъчна база, отразяваща реалистични условия на тестване.

Бъдещата работа включва разширяване на изследването към повече проекти и дефекти ($n \geq 100$) с цел валидиране на ML модели, изследване на влиянието на различни нива на информация в промпта върху успеваемостта, както и приложение на SmartTCP (Smart Test Case Prioritization) framework за приоритизиране на AI-генерирани тестове въз основа на предсказан индикатор P.

ЛИТЕРАТУРА:

[1] Just, R., Jalali, D., Ernst, M.D., Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In: Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA 2014). ACM (2014), 437-440.

[2] Xie, Z., Chen, Y., Zhi, C., Deng, S., Yin, J., ChatUniTest: a ChatGPT-based automated unit test generation tool. arXiv preprint arXiv:2305.04764 (2023).

[3] CodiumAI, Cover-Agent: An AI-Powered Tool for Automated Test Generation and Code Coverage Enhancement. GitHub repository (2024), https://github.com/Codium-ai/cover-agent.

[4] Tufano, M., Drain, D., Svyatkovskiy, A., Deng, S.K., Sundaresan, N., Unit Test Case Generation with Transformers and Focal Context. arXiv preprint arXiv:2009.05617 (2020).

[5] Rothermel, G., Untch, R.H., Chu, C., Harrold, M.J., Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering **27** (10) (2001), 929-948.

Ивайла Й. Пенчева
Шуменски университет „Епископ Константин Преславски“
ORCID ID: 0009-0009-7907-0127
E-mail: i.pencheva@shu.bg