Annual of Konstantin Preslavsky University of Shumen
Faculty of Mathematics and Informatics
vol. XXVII C, 2026, pp. 81 – 91, https://doi.org/10.46687/TZOE9636

APPLICABILITY OF REGRESSION TEST PRIORITIZATION TECHNIQUES IN MODERN SOFTWARE DEVELOPMENT[*]

IVAYLA Y. PENCHEVA

ABSTRACT: *Regression testing remains a fundamental part of software quality assurance, but its scalability challenges have become increasingly evident in modern CI/CD environments. Test Case Prioritization (TCP) addresses these challenges by reordering test cases to achieve faster fault detection when time and resources are limited. This paper explores three practically proven approaches — time-based, code coverage–based, and history-based prioritization. The comparative analysis shows that no single method is universally optimal; rather, the best choice depends on factors such as project maturity, available tools, and organizational priorities. Therefore, teams should aim for a context-aware strategy that balances simplicity, performance, and maintainability. The study concludes that adaptive prioritization powered by machine learning offers a promising path for future research and industrial adoption.*[†]

KEYWORDS: *Test Case Prioritization, Regression Testing, CI/CD, Code Coverage, Software Quality.*

2010 Math. Subject Classification: 68M15, 94C12

ПРИЛОЖИМОСТ НА ТЕХНИКИ ЗА ПРИОРИТИЗАЦИЯ НА РЕГРЕСИОННИ ТЕСТОВЕ В СЪВРЕМЕННАТА СОФТУЕРНА РАЗРАБОТКА

[*]This paper is (partially) supported by Scientific Research Grant RD-08-91/02.02.2026 of Konstantin Preslavsky University of Shumen

[†] *Бележка от автора: Части от текста са редактирани с помощта на инструмент с изкуствен интелект с цел езикова яснота и стилова консистентност. Авторът носи пълна отговорност за съдържанието и изводите.*

ИВАЙЛА Й. ПЕНЧЕВА

1. Въведение

Регресионното тестване е вид функционално тестване, фокусирано върху осигуряване на цялостната стабилност на софтуера. То е тестова дейност, която се извършва, за да се гарантира, че промените не нарушават съществуващото поведение на софтуера. Тестовите пакети са склонни да нарастват по размер с развитието на софтуера, което често прави твърде скъпо изпълнението на всички - те могат да достигнат до хиляди тестове, изискващи много време за изпълнение.

Проучени са редица различни подходи за максимизиране на стойността на натрупания тестов пакет: минимизиране, селекция и приоритизиране. Минимизирането на тестовия пакет елиминира излишните тестови случаи, за да намали броя на тестовете за изпълнение. Селекцията на тестови случаи идентифицира тези, които са релевантни за определен набор от скорошни промени. Приоритизирането на тестовете има за цел да ги подреди по такъв начин, че да максимизира скоростта на откриване на дефекти [4]. Приоритизирането предлага оптимален баланс между ефективност и запазване на тестовото покритие, което го прави особено приложимо в съвременните CI/CD среди.

За да се разбере пълната стойност на приоритизирането, важно е да се уточни кога се прилага регресионното тестване. То се изпълнява в няколко случая: след корекции на грешки – така се осигурява, че поправката в кода не влияе негативно на други функции. След добавяне на нови функции – за да се валидира, че новата функция работи правилно, без да нарушава някоя от старите. След подобрения или актуализации, както и след промени в конфигурацията - дори промените в хостинг средата или други компоненти могат да изискват регресионно тестване.

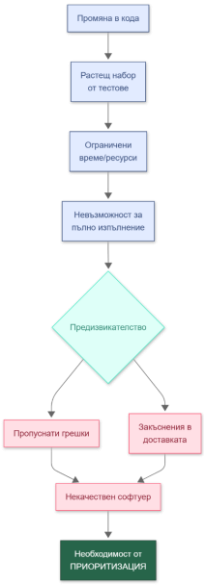
Значението на регресионното тестване е съществено, защото с него се откриват дефекти в ранен етап от разработката. Чрез ранното откриване на грешки, се намалява времето и разходите, свързани с отстраняването им по-късно в цикъла на

разработка. Освен това софтуерът остава надежден и функционален за потребителите, въпреки непрекъснатото му развитие с нови функции и актуализации.

Предизвикателството

Основното предизвикателство на регресионното тестване е неговата скалируемост. С развитието на софтуерния проект тестовите пакети нарастват непрекъснато - в индустриални системи те често достигат хиляди тестове с продължителност от часове до дни. Изпълнението на всички тестове при всяка промяна в кода става непрактично от гледна точка на време и ресурси, особено в съвременните CI/CD среди, където скоростта на доставка е критична. С всяка нова промяна възниква риск от регресия - нарушаване на съществуваща функционалност. Същевременно пълното изпълнение на регресионния пакет е невъзможно, поради времеви ограничения. Това създава дилема: как да се гарантира качеството без да се жертва скоростта на разработка? Приоритизацията на тестови случаи предлага решение на този проблем. Вместо да се изпълняват всички тестове или да се избират произволни подмножества, приоритизацията подрежда тестовете така, че дефектите да се откриват възможно най-рано. По този начин се постига оптимален баланс между изчерпателност на тестването и ефективност на ресурсите. В практиката изборът на конкретна техника за приоритизация се определя не само от теоретичната ѝ ефективност, но и от оперативна осъществимост, налични инструменти и интеграция в съществуващия работен процес.

Целта на тази статия е да анализира кои техники за приоритизация намират най-широко приложение в индустрията и да идентифицира практическите фактори, които определят техния избор.



Фигура 1: Схема на предизвикателствата, породени от нарастващия набор от регресионни тестове.

2. Най-разпространените техники в практиката

Въпреки че в академичните среди се проучват много сложни алгоритми, в индустриалната практика се налагат техники, които са лесно разбираеми, лесни за внедряване и дават бърз резултат.

2.1. Приоритизация по време за изпълнение

Тестовите случаи се подреждат в нарастващ ред на времето за изпълнение - най-бързите тестове се изпълняват първи. С широкото приложение на непрекъснатата интеграция (CI) в софтуерните проекти, което увеличава честотата на изпълнение на софтуерни компилации, изпълнението на всички тестове може да бъде отнемащо време и ресурси. За да се облекчи този проблем, се използват техники за приоритизиране на тестови

случаи, които подобряват регресионното тестване, за да се предостави ранна обратна връзка на разработчиците [2]. Простотата на този подход е едно от най-големите му предимства. Имплементацията се свежда до измерване и сортиране по време, което всеки разработчик може да реализира за минути. Приложимостта е много висока - техниката е подходяща за почти всеки проект, независимо от неговата архитектура. Особено ефективна е за "commit stage" в CI/CD pipeline, където се изисква бърза валидация и незабавна обратна връзка. Основният недостатък е, че техниката не открива непременно най-критичните дефекти първи. Бързите тестове не корелират с важността на тестваните компоненти. При дефекти в бавни модули, откриването им може да се забави значително.

2.2. Приоритизация, базирана на покритие на кода (code coverage)

Тестовете се подреждат според обема код, който покриват. Най-често се използва вариантът "допълнително покритие" (additional coverage), при който винаги се избира тестът, покриващ най-много все още непокрит код. Този подход гарантира максимално покритие на кодовата база с минимален брой изпълнени тестове. Техниката е базирана на конкретни данни за покритието на кода, което я прави лесно доказуема и обективна. Фокусът е директно върху променените части на системата и техните зависимости, което я прави много ефективна при таргетирано тестване. Добрата интеграция е друго важно предимство - съвременните инструменти за непрекъсната интеграция (CI) често поддържат инструменти за измерване на покритието, което улеснява внедряването. Непрекъснатата интеграция (CI) значително намалява проблемите с интеграцията, ускорява времето за разработка и съкращава времето за пускане на продукта. Тя обаче въвежда и нови предизвикателства за дейностите по осигуряване на качеството, включително регресионно тестване. Въпреки че различни подходи за приоритизиране на тестови случаи са се показали като

многообещаващи в контекста на регресионното тестване, трябва да бъдат разработени специфични техники, които да се справят с динамичния характер и времевите ограничения на CI [1]. Основният недостатък е изчислителната сложност - additional coverage е $O(n^2)$ алгоритъм, което може да създаде проблеми при много големи тестови пакети. Инструментацията на кода за измерване на покритие може да забави изпълнението с 15-30 процента.

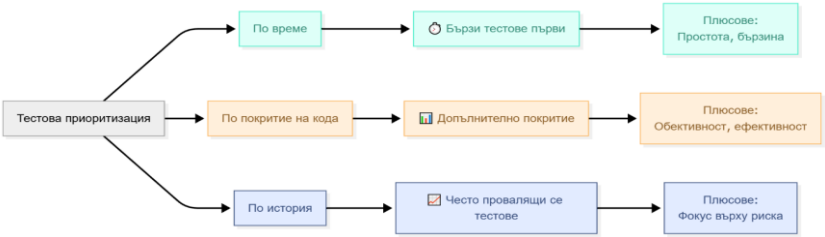
2.3. Приоритизация, базирана на история от изпълнения

През 2002 г. Ким и Портър предлагат първата техника за приоритизиране, базирана на история. Приоритизирането, извършено в техниката, се основава на исторически данни за изпълнение. Те показват, че историческата информация може да бъде полезна за намаляване на разходите и повишаване на ефективността на процеса на регресионно тестване. [5]

Същината на този метод е, че тестове, които в миналото са откривали повече дефекти или са се проваляли по-често, получават висок приоритет и се изпълняват първи. Техниката използва исторически данни за изпълнения, провали и открити дефекти, за да идентифицира "нестабилни" зони от кода. Основното предимство е фокусът върху риска - техниката се насочва директно към откриването на дефекти, вместо да разчита на структурни метрики.

Ефективността е друго важно качество - ако има качествен набор от исторически данни, тази техника често показва отлични резултати в откриването на дефекти. Адаптивността я прави още по-ценна - моделът може да се преизчислява периодично, за да отчита най-новите тенденции и промени в кодовата база.

Основният недостатък е зависимостта от данни - техниката изисква наличието на качествени исторически данни. За нови проекти или нови тестове възниква проблем, тъй като няма достатъчно информация за точна приоритизация.



Фигура 2: Таксономия на основните техники за приоритизация на регресионни тестове и техните предимства.

Практически фактори, влияещи на избора

Изборът на подходяща техника рядко се свежда до нейната теоретична ефективност. В практиката решението се определя от комплексно взаимодействие между организационни цели, технологична инфраструктура и характеристики на проекта. Организационните цели са първият критичен фактор. Ако приоритет е бързата обратна връзка в рамките на минути, приоритизацията по време е оптималният избор. Тази стратегия осигурява незабавна индикация за здравето на кода, което е от критично значение в среди с висока честота на промени. Ако целта е максимално откриване на дефекти преди пускане, приоритизацията базирана на покритие на кода или история предлага по-висока ефективност, макар и за сметка на по-дълго време за изпълнение. При фокус върху икономическа ефективност, балансираният подход комбинира техники според фазата на разработка - бързи тестове в commit stage и по-изчерпателни в pre-release валидация. Наличните данни и инструменти определят кои техники са практически осъществими.

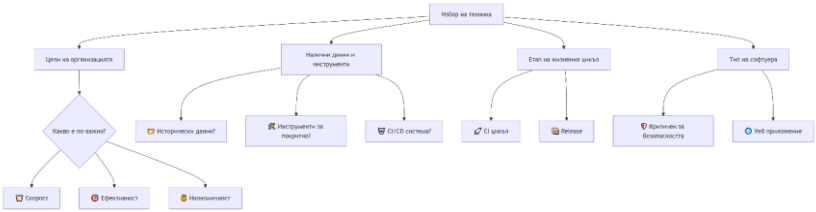
Приоритизацията базирана на история изисква система за проследяване на дефекти и значителна история на изпълнения - минимум 3-6 месеца качествени данни. Без такава инфраструктура техниката е неприложима.

Приоритизацията по покритие на кода зависи от наличието на инструменти за code coverage в CI/CD системата. Съвременните CI платформи като Jenkins, GitLab CI и GitHub Actions често поддържат такива инструменти "out of the box", което улеснява внедряването.

В случаите на нови проекти или ограничена инфраструктура, приоритизацията по време е най-достъпният начален подход, тъй като изисква само измерване на продължителността. Етапът на жизнения цикъл на разработка диктува различни стратегии. В рамките на CI цикъл, където се изисква обратна връзка за минути, се предпочитат бързи техники като приоритизация по време. Това позволява на разработчиците да получат незабавна валидация при всеки commit. Преди пускане до клиентите обаче могат да се приложат по-бавни, но по-изчерпателни техники, базирани на история или code coverage. В този етап времето не е толкова критично, а откриването на максимален брой дефекти е приоритет.

Типът на софтуера е друг важен фактор при избора. За критични за безопасността системи - медицински софтуер, финансови платформи, системи за управление на транспорт - се прилагат агресивни техники за максимално откриване на дефекти, често комбинирани подходи базирани на покритие и история. За уеб приложения с чести промени и по-нисък критичен риск, техниките даващи бърза обратна връзка са по-подходящи, за да се поддържа висока скорост на разработка.

Зрялостта на проекта също влияе на оптималния избор. Стари, монолитни системи с дългогодишна история на промени и натрупани дефекти се възползват максимално от приоритизация по история. Богатите исторически данни позволяват на ML моделите да идентифицират рискови зони с висока точност. От друга страна, нови микросървисни архитектури с ограничена история са по-подходящи за техники базирани на code coverage, които фокусират върху структурните характеристики на кода и промените в него.



Фигура 3: Дървовидна диаграма на ключовите практически фактори, влияещи върху избора на техника за приоритизация.

3. Заключение

В съвременната софтуерна разработка приоритизацията на регресионните тестове вече не е лукс, а необходимост. Динамичният бизнес климат и нарастващата сложност на системите изискват интелигентни стратегии за управление на тестовите пакети. Настоящото изследване демонстрира, че няма единна "най-добра" техника, която да отговаря на всички сценарии.

Трите най-широко приложими подхода - приоритизация по време, базирана на покритие на кода и базирана на история - предлагат различни баланси между простота, скорост и ефективност. Приоритизацията по време осигурява най-бърза обратна връзка и е универсално приложима, но не корелира с критичността на дефектите. Техниката базирана на покритие фокусира върху променените части на системата и е добре интегрирана в съвременните CI/CD инструменти, но изисква изчислителни ресурси. Приоритизацията по история показва най-висока ефективност при откриване на дефекти, но зависи критично от наличието на качествени исторически данни.

Успешната стратегия се крие в способността на екипите да диагностицират собствения си контекст - организационни цели, налична инфраструктура, тип на софтуера и зрялост на проекта. Практикуващите специалисти трябва да избират или дори да комбинират техники според конкретните си обстоятелства. В

много случаи хибридният подход, използващ бързи техники в commit stage и по-изчерпателни в pre-release валидация, предлага оптимален баланс. Успехът не се крие в избора на теоретично "най-добрата" техника, а в избора на най-подходящата за конкретния контекст.

Бъдещата насока на областта е насочена към използването на машинно обучение за персонализирана и адаптивна приоритизация. Интегрирането на изкуствен интелект и техники за машинно обучение революционизира процесите на приоритизация на тестове, позволявайки по-прецизно идентифициране на релевантните тестови случаи, като същевременно значително намалява времето за изпълнение. Чрез усъвършенствано разпознаване на модели и поведенчески анализ, съвременните системи демонстрират забележителни възможности за поддържане на цялостно тестово покритие, като същевременно оптимизират използването на ресурсите.

Внедряването на облачни и безсървърни архитектури допълнително подобрява тези възможности, позволявайки разпределени стратегии за тестване, които се мащабират ефективно с изискванията за разработка. Организациите, внедряващи тези сложни стратегии, съобщават за значителни подобрения в честотата на внедряване, използването на ресурсите и общата ефективност на разработката. Еволюцията на практиките за приоритизация продължава да се ускорява с нововъзникващите технологии, особено в области като интеграция на периферни изчисления и тестване ориентирано към микросървиси, обещавайки още по-голям потенциал за оптимизация в бъдещите практики за разработка на софтуер [3].

ЛИТЕРАТУРА:

[1] Bagherzadeh, M., Reinforcement learning for test case prioritization., IEEE Transactions on Software Engineering, 2021

[2] Pan, R., Test case selection and prioritization using machine learning: a systematic literature review, *Empirical Software Engineering*, 2022

[3] Kodithyala, S. K., Smart Test Selection in CI/CD: Optimizing Pipeline Efficiency, *Journal of Computer Science and Technology Studies*, 2025

[4] Yoo, S., Regression testing minimization, selection and prioritization: a survey, Software Testing, Verification and Reliability, 2013

[5] Singh, Y., Systematic Literature Review on Regression Test Prioritization Techniques, Informatica 36, 2012

Ивайла Й. Пенчева
ШУ "Епископ К. Преславски", ФМИ
i.pencheva@shu.bg