ТЕСТВАНЕ ПРИ ТРАДИЦИОННИТЕ И ГЪВКАВИТЕ ПОДХОДИ ЗА РАЗРАБОТКА НА СОФТУЕР

Теодора Николова, Стефка Толева-Стоименова, Велиян Димитров

Университет по библиотекознание и информационни технологии

Резюме: Настоящият доклад съдържа преглед на състоянието на софтуерното тестване. Обхваща видовете тестване, основните техники, които се прилагат за тестване на софтуерни приложения, жизнения цикъл на тестването и спецификите му в различните модели за разработване на софтуер. Изложен е основният терминологичен апарат в областта на софтуерното тестване, което е неизменна част от процеса на разработване на софтуерни продукти, както и ролята на специалистите, отговорни за тестването.

Ключови думи: тестване, софтуер, гъвкави, подходи, разработване.

Въведение

Софтуерната индустрия се развива динамично и изисква бързо доставяне на софтуерни продукти, които стават все по-сложни и се нуждаят от проверка на качеството. Тестването представлява процес на изследване на софтуера или части от него в контролирана среда с цел получаване на информация за недостатъците на софтуерния продукт и откриване на отклонения от изискванията. Различни са наименованията за специалиста, който тества софтуера, в зависимост от неговия опит и функция: софтуерен тестер, инженер по тестване, инженер по осигуряване на качеството (SQA engineer), анализатор на качеството (QA analytic) и др. Тестването се провежда в различни форми на всеки етап от жизнения цикъл на разработването на софтуера и тестерите са част от екипа, разработващ даден продукт.

Съвременната тенденция е все повече компании да преминават към гъвкавите подходи за разработка на софтуер, които са ориентирани към по-бързо доставяне на продукта на клиента, адаптивност по отношение на променящите се изисквания и високо качество. Гъвкавите технологии подменят фундаментално принципите на разработка и интегрират тестването на всички етапи.

Настоящият доклад представлява преглед на състоянието на

софтуерното тестване, типовете тестване и основните техники, както и на жизнения цикъл на тестването и мястото му в различните модели за разработване на софтуер. Проучването има за цел да се направи анализ на различните аспекти на процеса на тестване на софтуер и на ролята на специалистите по тестване в утвърдените и гъвкавите подходи за разработка на софтуер.

Методология на изследването

Методологията на изследването се основава на задълбочен документален преглед и сравнителен анализ на информацията относно процеса на разработка на софтуерни продукти, с фокус върху етапа на тестване и неговата роля при различните модели на разработка в традиционните и гъвкавите подходи, както и на систематизирането и обобщаването на събраната информация.

Видове тестване и използвани техники

Работата на тестерите е зависима от приложната област и спецификата на софтуера. Съществуват над 100 вида тестване, което показва колко разнообразна и предизвикателна е тази област. Ще бъдат споменати най-често използваните, без да се пренебрегва значимостта на всички останали.

В зависимост от спецификата на софтуера, който ще се тества, и редица фактори, като време, бюджет, опит на инженерите по тестването и др., се използват различни типове тестване, които могат да се прилагат на всяко от четирите нива [1], [2], [3]:

Тестване на ниво програмна единица (unit testing) – обект на тестването е един компонент (системна единица), неговата функционалност и комуникация с други компоненти.

Интеграционно тестване (integration testing) – тестват се по-големи структурни единици и подсистеми, като има три варианта:

- Големият взрив (big bang) – всички компоненти или системи са тестват едновременно за правилно интегриране, но по този начин трудно се открива причината за възникването на дефектите;
- от горе надолу (top-down) – първоначално се тества логиката на високо ниво, а компонентите на ниско ниво се тестват по-късно;
- от долу нагоре (bottom-up) – високото ниво на логика и най-сложните функционалности се тестват накрая, което се превръща в недостатък на този метод.

Системно тестване (system testing) се фокусира върху системата и нейните функции и характеристики като резултат от взаимодействието на всичките ѝ компоненти. На това ниво

причините за появата на дефекти най-често са неясни или липсващи системни изисквания, липсваща спецификация за правилно поведение на системата, пропуснати решения, непрегледани и неодобрени изисквания и/или грешна реализация на цялата система.

Тестване за приемане (acceptance testing) позволява включването на потребителя в процеса на разработка на софтуера. Този тип тестване се изпълнява в среда, близка до оперативната целева среда, от самите потребители, които предоставят на организацията, разработила софтуера, данни за инцидентите, възникнали със системата.

Поради това, че изчерпателно тестване не е възможно, тестерите разполагат с разнообразни техники, които помагат за оптимизирането на неговата ефикасност и ефективност. Най-популярните сред тях са: на черната кутия, на бялата кутия и на опита [2], [4].

Тестването по **метода на черната кутия** представлява непрозрачна техника за тестване и се извършва на базата на входните и изходните данни. Изследва само фундаменталните аспекти на системата и няма никаква или има малка връзка с вътрешната ѝ логическа структура. Независимостта от конкретната имплементация прави тестовите сценарии валидни, докато не се променят очакванията на потребителите. Затова тези техники са основа за създаването на автоматизирани тестове, които впоследствие спестяват много време и могат да бъдат използвани за регресионно тестване.

Тестването по **метода на бялата кутия** е прозрачна техника, която се основава на подробно изследване на вътрешната логика и структура на кода. То е подходящо най-вече за етапа на юнит тестовете, когато се гарантира, че всички пътища ще бъдат обходени поне веднъж и ще бъдат изпълнени всички логически решения.

Тестването **на базата на опита** на тестера се среща в два варианта: предположение за грешка (Error Guessing) и изследователско тестване (Exploratory Testing). При **метода за отгатване на грешка** силно влияние оказва опитът на тестера с подобни приложения или техни функционалности, с познати дефекти, откривани и отстранявани в миналото. Техническите му познания позволяват да предположи къде може да бъде допусната грешка в самия код, поради което този вид тестване понякога се нарича тестване по **метода на сивата кутия**.

В гъвкавите подходи често се използва изследователското тестване, при което специфицирането и провеждането на тестове се изпълняват едновременно. Определя се като неформална техника за тестване, при която тестерът активно контролира създаването на тестовите случаи. След като бъдат изпълнени, той използва информацията, която получава, за да създава по-добри тестове. В определен смисъл изходът е непредсказуем, но тестването не е хаотично, а подчинено на т.нар. *план за изследователско тестване*, съдържащ по точки какво и кога ще се тества, времетраене, кой ще тества, цели на тестването, както и резултата от тестването и потенциалните дефекти.

Тестването, базирано на опита, често е предпочитано, защото дава възможност за повече креативност, особено когато липсва спецификация или техническа документация, няма време за анализ и дизайн, както и за разработване на тестови случаи, но също и тестерите имат достатъчно опит в областта. Същевременно по-често тестването, базирано на опита, се съчетава с останалите техники, за да не се разчита изцяло на субективната преценка на тестера.

Изброените по-горе техники може да бъдат включени изолирано или в комбинация в дейностите, свързани с тестването. В зависимост от това какви са целите на тези дейности и към кои аспекти на компонента/системата е насочено тестването, може да се дефинират различни типове тестване.

Функционалното тестване използва непрозрачна техника и може да се изпълнява на всички нива на тестване, като се следваъ стъпките: идентифициране на входните данни, идентифициране на изходните данни, изпълняване на тестовите случаи, сравняване на резултатите. Тестовете се създават на базата на функционалните изисквания [1], [2].

Нефункционалното тестване показва как се държи приложението от гледна точка на надеждност, използваемост и ефективност на работата. Извършват се тестове за изпълнение, за натоварване, за обем, за сигурност, стрес тестване и др. [1]. **Тестването за потвърждаване** се извършва след отстраняването на дефект, за да се потвърди, че дефектът наистина е отстранен. **Регресионното тестване** в определен смисъл е свързано с тестването за потвърждаване. Целта е да се провери дали промяната на кода след отстраняване на дефекти не засяга съществуващата функционалност на продукта. **Тестването, базирано на анализа на риска**, включва оценка на риска въз основа на сложност на софтуера, критичност на потребителя, честота на

използване на дадената функционалност, възможни области с дефекти и др. **Тестването при гъвкавите подходи** се фокусира върху клиентите и доставянето на работещ софтуер [1], [2].

Жизнен цикъл на разработването и на тестването

Разработването на всеки софтуерен продукт преминава през няколко основни етапа, определящи т.нар. жизнен цикъл на разработване на софтуер (Software Development Life Cycle, SDLC) – анализ на изискванията, дизайн, разработване, тестване, внедряване и поддръжка. Дейностите, свързани с тестването, заемат своето място в жизнения цикъл на самата разработка, но същевременно тестването преминава през собствен жизнен цикъл (Software Testing Life Cycle, STLC), както е посочено в таблица 1. В него могат да се разграничат етапите: анализ на изискванията, планиране, анализ и дизайн, имплементация, изпълнение на тестовете, създаване на отчети за изпълнението на тестовете и откритите дефекти, заключителни дейности [5].

Таблица 1. Жизнен цикъл на разработка (SDLC) и тестване на софтуер (STLC)

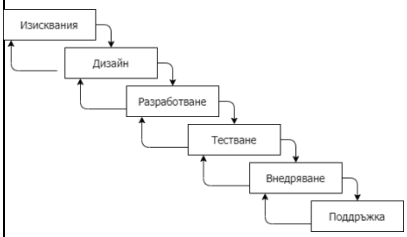
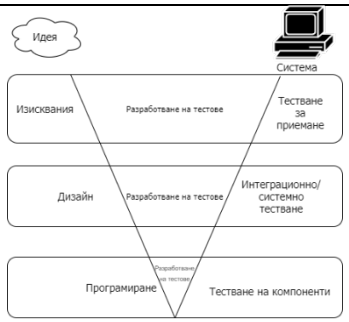
a) SDLC	b) STLC
	

Съществуват използвани в практиката модели, някои от които са показани в таблица 2: водопаден (a),V-модел (b), итеративен модел (c) [1], [6], [7].

При **водопадния модел** всеки етап протича отделно от останалите. От гледна точка на тестването водопадният модел има предимството, че всички етапи са дефинирани и в резултат екипът по качеството разполага с ясно определени и документирани изисквания. От друга страна, тестването протича след имплементацията и ако бъде открит съществен недостатък, цената на отстраняването на проблема може да е доста висока.

V-моделът, подобно на водопадния, се характеризира с ясно разграничени етапи, но на всеки етап от разработването на софтуера съответства етап от процеса на тестване. По този начин се увеличава вероятността да бъдат открити дефекти в ранните етапи на разработката.

Таблица 2. Модели на жизнения цикъл на разработката на софтуер

<table>
<tr><td>a) Водопаден модел</td><td>b) V-модел</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">c) Итеративни модели</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

При **итеративните модели** разработването на продукта се разбива на малки стъпки на базата на изискванията, подредени по значимост, а в края на всяка итерация на клиента се предоставя част от функционалността. Класически примери за итеративни модели са Моделът на бързото разработване (RAD, Rapid Application Development), Рационалният унифициран процес (RUD, Rational Unified Process) и гъвкавите подходи за разработване (Agile). Предимство на този модел за разработване на софтуер е, че възможно най-рано клиентите разполагат с работеща функционалност, която може постепенно да се разширява. На всяка итерация екипът по разработката получава обратна връзка от клиента, което намалява риска, дава възможност за промяна на изискванията в хода на работата и позволява постоянното подобряване на продукта. От друга страна, при итеративните модели се изискват значителни ресурси по отношение на екип, време, технологии и др.[2]

Тестване при гъвкавите подходи

Гъвкавите подходи за разработване на софтуер се основават на т.нар. Манифест за гъвкавите подходи (Agile Manifesto, 2001), според който фокус се поставя върху потребителя и взаимодействието между заинтересованите страни; процесите следват промените, а не предварително заложен план.[1]

Гъвкавите подходи използват специфични модели за разработване на софтуер: **екстремно програмиране**, **скръм** и **канбан**, като последните два са по-скоро методи за управление на проекти. Целта им е да се дефинират и организират дейностите, свързани с разработването на продукта, да се определят редът на елиминиране на потенциалните рискове, начините за справяне с проблемите и комуникацията между членовете на екипа [8].

Екстремното програмиране се базира на пет основни ценности при разработването на софтуер: комуникация, простота, обратна връзка, смелост и респект. Изпълняват се само предвидените изисквания. Процесите се адаптират към проекта, а не той към модела. Тестването е ключова дейност в този модел, доколкото в основата му стои т.нар. Test-Driven Development, т.е. първо се разписват тестовете, а след това се преминава към имплементацията, за да се подсигури правилната работа на функционалността [9].

Скръм (Scrum) представлява итеративно-инкрементален подход за управление на сложни проекти. Позволява насочването

на фокуса върху ефективното сътрудничество между членовете на екипа и прозрачността в тяхната работа. Проектът се разбива на отделни итерации с фиксирана дължина (спринтове), като в края на всеки спринт се предоставя завършена функционалност [10].

В основата на **канбан** е оптимизацията на работния процес чрез визуализация на задачите по разработване или тестване на т.нар. канбан дъска. При този метод съществува ограничение по отношение на задачите, които могат да се извършват едновременно, и се оптимизира времето за изпълнение на задачите.[3]

На основните принципи на Agile се базира и тестването при гъвкавите подходи за разработване. В таблица 3 са посочени някои основни фактори, които разграничават гъвкавите и традиционните подходи и процеса на тестване при тях [11].

Гъвкавите подходи при разработване на софтуер се разглеждат като вид итеративни модели, доколкото разработването на функционалността и тестването се осъществяват на спринтове, които обаче са изключително кратки – обикновено до две-три седмици. Понеже итерациите са последователни, това предполага, че разработването на функционалността и тестването се осъществяват едновременно, преди да започне следващата итерация. Следователно тестването не е изолирана, самодостатъчна дейност, а е интегрирано в цялостната дейност на екипа [12].

Таблица 3. Сравнение на тестването при традиционните и гъвкавите подходи

Фактор	Гъвкав	Традиционен
Екипна работа	Интегрирани в екипа по разработка.	Отделен екип от тестери.
Процес	Подготовката за тестване започва с потребителски истории и тестването продължава в следващите етапи на разработката. Тестването е итеративно и най-често автоматизирано.	Извършва се на няколко етапа: подготовка на теста, изпълнение, документиране и комуникация чрез документация. Тестването се осъществява след приключването на разработката.
Комуникация	Непрекъсната комуникация със заинтересованите страни.	Рядка комуникация с клиентите, екипът по разработка и заинтересованите страни.
Планиране	Общо очертаване.	Прецизно документиране.
Цена	По-ниска цена и минимално завишаване при промени.	По-висока цена и голямо завишаване при промени.

Фази	Зависи от големината на проекта.	Множество тестови фази.
Изисквания	Могат да се променят в хода на разработката. Адаптивност, промените не забавят проекта.	Ясно дефинирани от началния етап на жизнения цикъл на разработката. Промените много забавят.

Използването на гъвкав подход означава, че тестването се извършва непрекъснато, по време на целия процес на разработка, и включва разработчици, тестери и клиенти на почти всеки етап. Поддържането на непрекъсната обратна връзка подпомага процеса на разработка, тъй като екипът по тестването не е принуден да изчаква цялостната разработка, за да установи дефекти. В таблица 4 са систематизирани някои основни преимущества и слабости на тестването при гъвкавите подходи.[4]

Таблица 4. Предимства и недостатъци на тестването при гъвкавите подходи

Предимства	**Недостатъци**
Задачите се разбиват на малки итерации (от простото към сложното).	Лошо планиране на ресурсите по тестването поради непрекъснатата промяна на изискванията.
Гъвкавост и адаптивност.	Пренебрегване на документацията поради кратките спринтове.
Ефективност и ефикасност на тестването от експертен екип.	Изгубване на идеята за цялото, което може да доведе до появата на неконсистентен продукт и проблеми със системното тестване.
Продуктът се разработва на първо място през призмата на крайните потребители.	Разфокусиране на QA екипа поради липсата на установени процеси, защото се залага на неговата самоорганизираност, както и на тази на екипа като цяло.
Минимална работа по документацията, залага се на преизползването на ресурсите.	Разход на време, често се отстраняват незначителни проблеми, затова може да се увеличават регресионните бъгове, а това да забави изпълнението на проекта.

Изводи/Дискусия

Обзорът на най-популярните модели, респективно на гъвкавите подходи, и мястото на тестването във всеки от тях показва, че тестването е необходим етап от жизнения цикъл на софтуера.

При гъвкавите подходи акцентът не е върху отделните аспекти от жизнения цикъл на проекта, а по-скоро се обръща внимание на цялото (whole-team approach) и всеки член на екипа е еднакво отговорен за успеха на проекта. В определен смисъл това означава, че всички участват активно, например в изясняването на изискванията, както и, от друга страна, че не само тестерите, но и програмистите и клиентите имат роля в тестването: например разработчиците пишат юнит тестове, инженерите по тестването се занимават с интеграционно и системно тестване, потребителите са ангажирани с приемно тестване и т.н. Например при екстремното програмиране е възможна и съвместна работа между програмист и тестер, както и последният може да играе ролята на инструктор за екипа, да въведе в продукта нови членове на екипа или да ги запознае с работата по осигуряване на качеството.

В резултат на гореизложеното може да обобщим, че всеки етап от SDLC, включително тестването, е еднакво значим за качеството на софтуера. Подобряването на процесите в STLC неизбежно ще допринесе за подобряването на процесите в SDLC, което ще повиши и удовлетвореността на потребителите.

Заключение

Софтуерното тестване в гъвкавите подходи за разработка на софтуер представлява синергична дейност, в която теоретичните знания и техническите и психологическите похвати взаимно се допълват и усилват. Колкото и добри да са прилаганите модели за работа, управление и тестване, те са безсилни, ако няма мотивирани специалисти с професионални и психологически познания и умения, които да ги прилагат.

Във връзка с това специалистът по тестването трябва да познава добре всички етапи от жизнения цикъл на разработването на софтуер и на тестването, видовете тестване и практиките, свързани с осигуряването на качеството на продукта, както и да осигурява успешното сътрудничество между заинтересованите страни. Също така трябва да притежава аналитично мислене, комуникативни умения, умения за справяне с проблеми, управление на времето и други важни умения за справяне с комплексни софтуерни проекти.

В по-широк аспект всичко това означава, че при софтуерното тестване като цяло конвергенцията на теоретично, техническо и психологическо е съществено условие за подобряването не само на качеството на продукта, но и на процесите в екипа. Това съвместно функциониране на различни модели, метрики за управление на дефекти и типове тестване може да доведе до трансформирането и еволюирането им или дори до създаването на нови, с което е по-сигурно творческото постигане на значим и по-добър резултат.

Бележки

[1] **Beck**, K. et al. Manifesto for Aile Development, 2001. <https://agilemanifesto.org/> (5.05.2023).

[2] **Iterative** Model: Advantages and Disadvantages, 2019. <https:// www.professionalqa.com/iterative-model> (5.05.2023).

[3] **Lynn**, R. How to Use Kanban in Software Testing, Planview. https://www.planview.com/resources/articles/kanban-software-testing> (7.05.2023).

[4] **Rana**, K. Agile Testing – Features, Methods, Pros and Cons, Art Of Testing. https://artoftesting.com/agile-testing (8.05.2023).

References/Литература

1. **Dimitrov**, V. Testvane na softuer. Bazovi kontseptsii v testvaneto na softuer. Sofia: Avangard Prima, 2017.

[**Димитров**, В. Тестване на софтуер. Базови концепции в тестването на софтуер. София: Авангард Прима, 2017.]

2. **Black**, R. Advanced Software Testing. Vol. 1. 2nd Edition. Guide to the ISTQB Advanced Certification as an Advanced Test Analyst. Rocky Nook, 2016. https://doi.org/10.1016/j.procs.2022.07.116.

3. **Jovanavic**, I. Software testing methods and techniques. – In: *IPSI BGD Internet Research Society*, vol. 5(1), January 2009, pp. 30 – 41.

4. **Ehmer**, M., F. **Khan**. A Comparative Study of White Box, Black Box and Grey Box Testing Techniques. – In: *International Journal of Advanced Computer Science and Applications*, vol. 3(6), 2012, pp. 12 – 15. doi:10.14569/IJACSA.2012.030603.

5. **Choubisa**, R., P. **Dalal**. Software Testing: A Review. – In: *International Journal of Engineering Research & Technology (IJERT)*, vol. 2(11), November 2013, pp. 676 – 679.

6. **Seema**, S., S. **Kute**, D. **Surabhi**, A. **Thorat**. A Review on Various Software Development Life Cycle (SDLC) Models. Vol. 3, 2014, pp. 2320 – 5156.

7. **Hrabovská**, K., B. **Rossi**, T. **Pitner**. Software Testing Process Models Benefits & Drawbacks: A Systematic Literature Review. Software Ingenering, 2019.

8. **Abrahamsson**, P., O. **Salo**, J. **Ronkainen**, J. **Warsta**. Agile software development methods: Review and analysis. VTT publication 478,

Espoo, Finland, 2002, 107 p.
http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf.
9. **Ariza**, H. M., F. **Silva**, L. **Contreras**. Descriptive Analysis of the Agile Methodology Extreme Programming (XP) for its Implementation in Software Development. – In: *International Journal of Engineering Research and Technology*, vol. 14(10), 2021, pp. 999 – 1004.
10. **Tuomikoski**, J., I. **Tervonen**. Absorbing Software Testing into the Scrum Method. 10th International Conference Product-Focused Software Process Improvement (PROFES), Finland, 2009.
11. **Najihi**, S., S. **Elhadi**, R. **Ait Abdelouahid**, A. **Marzak**. Software Testing from an Agile and Traditional view. – In: *Procedia Computer Science*, vol. 203, 2022, pp. 775 – 782.
12. **Stolberg**, S. Enabling Agile Testing through Continuous Integration Agile Conference. Chicago, USA, 2009, pp. 369 – 374.

За авторите

Теодора Николова е докторант в катедра „Компютърни науки“. През 2022 г. завършва УниБИТ като магистър по специалността „Софтуерно инженерство“. Интересите ѝ са свързани с областта на тестването на софтуер.

За контакт с автора: t.nikolova@unibit.bg

Стефка Толева-Стоименова е доцент в катедра „Компютърни науки“ на УниБИТ. Завършила е Техническия университет – София, а през 2011 г. получава докторска степен в УниБИТ. Нейните публикации и изследователски интереси са в научните области информатика, наука за информирането и наука за данните.

За контакт с автора: s.toleva@unibit.bg

Велиян Димитров е професор в катедра „Компютърни науки“ на УниБИТ. Завършва Шуменския университет „Св. Константин Преславски“ като магистър по математика. През 2014 г. защитава дисертационен труд в сферата на информационната сигурност в УниБИТ. Интересите и изследванията му са в областта на киберсигурността и интегралната защита на големи системи от ИКТ.

За контакт с автора: v.dimitrov@unibit.bg

TESTING IN TRADITIONAL AND AGILE SOFTWARE DEVELOPMENT

Teodora Nikolova, Stefka Toleva-Stoimenova, Velian Dimitrov

University of Library Studies and Information Technologies

Abstract: This paper provides an overview of the state of software testing. It covers the types of testing, the basic software testing techniques, testing life cycle and its specifics in different software development models. The fundamental concepts in software testing, which is an invariable part of the software products developing process, as well as the role of the specialists responsible for testing, have been explored.

Keywords: testing, software, agile, approach, development.

About the Authors

Teodora Nikolova is a PhD student in Computer Science Department at ULSIT. In 2022, she graduated with a master's degree in Software Engineering from ULSIT. Her main research interests are related to software testing.

To contact the Author: t.nikolova@unibit.bg

Stefka Toleva-Stoimenova is an Associate Professor in Computer Science Department at ULSIT. She has obtained her MSc degree from the Faculty of Automation and System Design, Technical University – Sofia. In 2011, she received a PhD degree from ULSIT. Her publications and main research interests are in the field of Informatics, Informing Science, and Data Science.

To contact the Author: s.toleva@unibit.bg

Velian Dimitrov is a Professor in Computer Science Department at ULSIT. He has obtained her MSc degree in Mathematics from *Konstantin Preslavsky* University of Shumen. In 2014 he received a PhD degree from ULSIT. His research interests and publications are in the field of cybersecurity and integrated cyberprotection of large systems of ICT.

To contact the Author: v.dimitrov@unibit.bg