16. Управление на качеството на софтуерни приложения. Тестване на софтуер. План за тестване и тестови сценарии.

Една от основните характеристики на софтуерното инженерство като дял от бизнеса е разработката на КАЧЕСТВЕН софтуер. Качеството на софтуера най-общо казано е съответствието между създадения продукт или услуга и нуждите и очакванията на потребителя. Оценяването на качеството на даден софтуер се осъществява посредством моделите на качество.

1. Модели на качеството (класически и съвременни).

Качеството представлява композиция от много характеристики.
Разработените модели представляват описание на определен
набор от тези характеристики и взаимовръзките между тях.

1.1. Класически модели

- Тези модели се фокусират върху качеството на крайния **продукт** и във връзка с това задават ключовите **фактори** за качеството от гледна точка на потребителите.
- Факторите на качеството са абстрактни и не могат да бъдат директно измервани. За целта се прави декомпозиция на факторите до **атрибути**, които могат да бъдат директно измервани.
- По своята същност тези модели са **йерархични** модели.

1.1.1. Модел на McCall

Обхваща три обособени области в употребата на софтуерния продукт:

1.1.1.1. **Работа с продукта**.

Извършва се декомпозиция до следните фактори:

- Лекота за използване /Usability/
- Интегритет /Integrity/
- Ефективност /Efficiency/
- Коректност /Correctness/
- Надеждност /Reliability/

1.1.1.2. **Ревизия на продукта**. Факторите са:

- Възможност за поддръжка /Maintainability/
- Възможност за тестове/Testability
- Гъвкавост /Flexibility/

1.1.1.3. **Промяна на продукта** – фактори:

- Възможност за повторна употреба (Reusability);
- Възможност за преносимост /Portability/;
- /Interoperability/

Всички тези фактори се разбиват до съответното множество от измерими атрибути, наричани също така **критерии за качество**.

1.1.2. Модел на Boehm

1.2. Съвременни модели

1.2.1. ISO 9126

Модел за качество на продукт. Изхожда от модела на Маккол и най-приложимият към настоящият момент за оценка качество на продукт.

Функциониране /Functionality/

- Приложимост /Suitability/
- Прецизност /Accuracy/
- Сигурност /Security/
- /Interoperability/

Надеждност /Reliability/

16. Управление на качеството на софтуерни приложения. Тестване на софтуер. План за тестване и тестови сценарии.

- Степен на развитие /Maturity/
- Допустимост по отношение на откази /Fault-tolerance/
- Възможност за възстановяване /Recoverability/

Лекота за използване /Usability/

- Интуитивен /Understandability/
- Лесен за усвояване /Learnability/
- Лекота на осъществяване на операциите /Operability/

Ефективност /Efficiency/

- Бързодействие /Time behavior/
- Употреба на ресурси /Resource behavior/
- Възможност за анализи/Analyzability/
- Възможност за промени /Changeability/
- Стабилност /Stability/
- Възможност за тестване /Testability/

Възможност за преносимост /Portability/

- Възможност за адаптация /Adaptability/
- Възможност за инсталация /Installability/
- Съответствие /Conformance/
- Заменяемост /Replacebility/

1.2.2. NASA - SATC Software Quality Model

1.2.2.1. Същност

Този модел е насочен както към качеството на крайния продукт, така и към качеството на процеса по разработка на софтуера.

- дефинира множество от цели, които са свързани както с факторите за качество на софтуерния продукт, така и с динамиката на тяхното развитие по време на жизнения цикъл на проекта.
- съдържа процесно-ориентирани индикатори за качество както и класическите индикатори за продукт
- обвързан е силно с управлението на риска в проекта.

1.2.2.2. Цели на модела

- Качество на изискванията /Requirements Quality/;
- Качество на продукта{Кода} /Product(Code) Quality/;
- Ефективност на имплементацията (Implementation Efficiency);
- Ефективност на тестването (Testing Efficiency);

Целите се оценяват с дефинирано множество от атрибути, които са измерими **в рамките на жизнения цикъл** на проекта и дават нужната информация.

2. Класификация и видове тестове в зависимост от готовността на софтуерното решение - тестване на софтуерна единица, тестване на модул, интеграционно тестване на модули, тестване на система, потребителски тестове за приемане на системата).

Тестването е процес на изследване на софтуерните системи, чиято цел е да провери дали те удовлетворят потребителските изисквания и нужди. В хода на изследването могат да се открият грешки и несъответствия.

В зависимост от етапа на разработка на софтуерния продукт се определят следните типове тестове.

Тестов сценрий – последователност от потребителски стъпки, които реализират даден бизнес сценарий. Тестовият сценарий се състои от предпоставки, входни данни за изпълнение, дефинираната последователност от елементарни потребителски стъпки и очаквани резултати. При създаването на пакет от тестови сценарий се цели пълно

покриване на всички възможни ситуации, с колкото се може по-малък брой и по-прости сценарии.

2.1.Тестване софтуерна единица (Unit testing)

Софтуерна единица е термин, който носи значението на най-малката компилируема единица на софтуерната система.

Тестването на софтуерни единици обикновено се извършва от самите програмисти. Изисква детайлно познаване на дизайна и кода на модулът, в който попада единицата. При тестването на софтуерни единица предварително се дефинират статично входни параметри, с които ще се тества единицата и съответните изходи, които се очакват като резултат при изпълнението на тестовете. Тези тестове се пускат многократно в процеса на разработка на софтуер и гарантират, че съответната единица работи коректно.

Тестовете се базират на техническата спецификация и структурата на кода. Прилага се подхода на бялата кутия.

2.1.1. Помощни средства

Тестването на единици обикновено се осъществява с помощта на специални среди и инструменти за тестване.

- Такова помощно средство е **stub**-ът, който се ползва като заместител на програмни единици, с които комуникира единицата, която е обект на тестването. Най-често с оглед на това да тестваме единицата в изолация от другата част от системата се създават въпросните **stub**-ове.
- Друго помощно средство е **test driver**-а, който изпълнява функционалностти на дадена единици с цел да я изтества.

2.1.2. Подходи за unit testing

- **Top-down unit testing**
  Конструира се йерархия/дърво от единици, като най-отгоре е единицата, която ползва други единици. В началото ползваните единици се заместват със stub-ове и постепенно вместо stub-ове в тестовете участват реалните единици. Този подход е удачен за тестване при интеграция на изтествани вече единици.
- **Bottom-up unit testing**
  Тестването започва от най-ниските нива на йерархията от зависимости между програмни единици. След като се изтества дадена единица се отива една стъпка нагоре в йерархията. При този подход се ползват test driver-и, но не и stub-ове.
- **Isolated unit testing**
  Предимствата на изолираното тестване на единици е, че ако възникне грешка, веднага може да се каже в коя програмна единица е проблемът. Този подход в тестването е нужен, за да се използват другите подходи – bottom-up и top-down.

2.2.Модулно тестване (Module testing)

При модулното тестване се изследват функционални единици на софтуерната система. Тестващият трябва да може да се ориентира в самия код. Ако липсва пълна яснота по отношение на програмните единици в модула и тяхната йерархия, този тест се нарича **grey box** – свързва се с лимитираната видимост на кода. По-често метод за тестване на модули е методът на **бялата кутия,** при който тестващият има на разположение кода и го разбира.

Тестовете се извършват от програмисти и тестери. Необходимо е да се състави тестов план, включващ тестови сценарии, входни данни и очаквани резултати. Грешки, открити на този етап от разработката на софтуерното решение, са от висока важност. Обикновено се документират в Bug-Tracking системата.

16. Управление на качеството на софтуерни приложения. Тестване на софтуер. План за тестване и тестови сценарии.

2.3.Интеграционно тестване на модули (Integration testing)

Съвместното тестване на модули се нарича интеграционно тестване. Интеграционите тестове изследват как два или повече модула работят заедно. Този тип тестове могат да се разделят на две основни групи:

- С ниско ниво на интеграция (два или три модула)
- С високо ниво на интеграция (4+)

Интеграционното тестване най-често ползва white-box testing подхода с оглед на това да има видимост върху евентуални конфликтни зони между отделните модули. Подобни конфликтни зони, където най-често се откриват грешки са **интерфейсите, използваните работни области в паметта и базата данни**.

2.4.Системно тестване (System testing)

Системното тестване стартира едва след като всички функционални и технически изисквания на клиента са имплементирани. Включва различен набор от тестове в зависимост от спецификата на проекта. Задължително се тестват **всички възможни бизнес сценарии**, **сигурността** на системата, **надеждността** й и **възможността за възстановяване** след авария.

- Алфа тестове – след реализация на спецификациите
- Бета тестове – след алфа тестовете и при доказано стабилно поведение се стартира бета тестинг, чийто извършител са реални потребители
- Финални /release/ тестове – последни тестове, не се предвиждат никакви повече промени в приложението

2.5.Потребителски тестове за приемане на системата (User-acceptance testing)

- Обучение на потребителите
- Съвместни прегледи
- Тестване
- Анализ на резултатите
- Протокол за приемане на системата
- Описание на процедурата за управление на грешки

3. Подходи за тестване - бяла кутия, черна кутия, сива кутия.

3.1.Бяла кутия

При този подход за тестване, известен още като **white-box testing** и **glass-box testing**, тестващият има пълен достъп до кода на програмата и го разбира. По този начин тестващият има представа каква част от кода биват покрити от неговите тестове (**coverage of code statements, branches**). Този подход е приложим от хора с поне базови програмни и алгоритмични умения.

3.2.Черна кутия

При този подход тестващият няма никакъв достъп до кода на програмата. Той ползва единствено техническа спецификация на системата, за да създаде пакет от тестове с валидни и невалидни входни данни, с който пакет да верифицира и валидира коректността на системата.

3.3.Сива кутия

При този подход тестващият има лимитирана видимост върху софтуера. Най-често тази видимост е върху дизайнът на модула/ системата и/или върху интерфейсите на дадени модули или други програмни единици.

16. Управление на качеството на софтуерни приложения. Тестване на софтуер. План за тестване и тестови сценарии.

4. Класификация и видове тестове, свързани с функционални и технически изисквания към системата - функционално, регресионно, тестване на производителност, тестване на сигурност, други.

4.1. Функционални тестове

Тези тестове целят да покрият колкото се може по-голям процент от всички бизнес процеси и сценарии. За целта се ползва black-box тестване, при което тестерът не трябва да познава кода, а да симулира реалното поведение на потребителите. Работи се с набор от валидни и набор от невалидни данни. При едните програмата трябва да работи коректно, а при другите да уведомява адекватно потребителя за възникнали грешки.

4.2. Регресионни тестове

Това тестване се изпълнява винаги след извършването на значителни промени по кода с цел да се провери дали промените в една област не са довели до грешки в друга. При компаниите, които държат на качеството на своите продукти регресионни тестове се правят при всеки build, което най-често става автоматично със съответните помощни средства.

4.3. Smoke/Sanity тестове

Тези тестове целят да покрият само основната функционалност на даден софтуер. Ако те минат, тогава се пускат целият пакет от тестове.

4.4. Тестове за производителност /Performance/

Служи за оценка поведението на системата при реални натоварване. Критерии за качество са response time, transaction rates. Симулира се паралелната работа на множество потребители.

- Load testing – тества системата при очакваното натоварване
- Stress testing – тества системата при свръхнатоварване

4.5. Тестове за сигурност /Security/

- Приложно ниво на сигурност (потребителски достъп)
- Системно ниво на сигурност

4.6. Тестове за удобство за работа /Usability/

Тези тестове са субективни, зависят от характера на съответните потребители. Целят да проверят дали интерфейсът на програмата е ясен и разбираем за потребителите. Тестовете обикновено включват наблюдения върху работата на група потребители със системата.

4.7. Тестове за възстановяване след срив /Backup and recovery tests/

Този тип тестове има за цел да изследва поведението на системата при евентуален неин срив. Проверяват се вътрешните й механизми за съхранение на текущото състояние и механизмите за възстановяване на предишно състояние. Целта е да се удостовери, че при изключителни ситуации няма да има загуба на данни.

Тестването протича като система се подлага на извънредни ситуации – хардуерна повреда, входно-изходни грешки, след което се проверява **реакцията на системата, времето за възстановяване** и евентуално количеството **загубени данни**.

16. Управление на качеството на софтуерни приложения. Тестване на софтуер. План за тестване и тестови сценарии.

5. Техники за оптимизация на функционални тестови сценарии - клас на еквивалентност, анализ на граничните стойности, таблица за вземане на решение).

5.1.Класове на еквивалентност

5.1.1. Същност

Работи се с определени класове на еквивалентност. Входните елементи се обединяват в групи, категоризирани според резултатите или изходното поведение на програмата. Ако вход А и вход Б водят до едни и същи резултати, то тези два елемента попадат в един и същ клас на еквивалентност.

5.1.2. Приемущества

Разделянето на класове на еквивалентност повишава ефективността на процеса по тестване, намалявайки броя на реализираните тестови сценарии без това да се отрази върху коректността на тестването.
На практика избягваме изпълнението на излишни тестове, но запазваме пълнотата и коректността на тестовия процес.

5.1.3. Приложение

Подобно разделяне на едно множество на различни класове на еквивалентност е приложимо не само спрямо входа, а и спрямо изхода или други характеристики на работата със системата.
Единственото нужно е да се определи **Релация на еквивалентност** спрямо дадените множества (вход, изход)

- Weak equivalence class testing – по една променлива за всеки клас на еквивалентност;
- Strong equivalence class testing – пълно комбиниране на класовете на еквивалентност;

5.2.Анализ на граничните стойности

Това е най-разпространената техника за функционално тестване. Тя се фокусира върху изследване на областта на входните параметри. Идеята на тази техника е за всяка променлива да се разгледат 3 стойности – минимална, близка до минималната (но във вътрешната част на интервала от допустими стойности), номинална, близка до максималната (но във вътрешната част на интервала от допустими стойности) и максимална.

Основният принцип, който се използва, е **Принципът на единичната грешка** – > причината за дефект в някакъв продукт много рядко се дължи на едновременното допускане на две различни грешки. В резултат на този принцип, тестовите случаи се построяват по следния начин: *във всеки тестов случай точно една от променливите се взима с някоя от граничните стойности (тъй като възможността за грешки при тях е най-голяма) и се комбинира с номиналните стойности на останалите променливи*.

Техниката на граничните стойности е подходяща за числови променливи, чиито допустими стойности са зададени с интервал, когато входните променливи са независими една от друга и когато променливите имат характер на физически величини – температура, скорост, ъгъл.

Тестването за устойчивост е разширение на тестването с гранични стойности – добавят се още две възможни стойности – близо до минималната, но извън интервала

от допустими стойности, и близо до максималната, но извън интервала от допустими стойности. Целта на тези тестови сценарии е да се провери дали системата е стабилна, когато с нея се работи неправилно.

Разновидност на тестването с гранични стойности е **Тестването в най-лошия случай** (worst case testing). При него не се спазва Принцип за единичната грешка и се изследва поведението на системата, когато повече от една променлива има стойност, различна от номиналната.

5.3.Таблица за вземане на решение

При тази техника се обвързват изискванията към системата, които съдържат различни логически условия, с вътрешния й дизайн. На базата на анализ на спецификацията се идентифицират различните входни състояния /комбинация от възможни входове/ и обработките, през които трябва да премине системата в зависимост от тях. Възможните условия се представят като комбинация от False/True входни стойности.
Описват се и всички допустими действия, които могат да се асоциират към условията. Таблицата дава зависимостта за изпълнението или неизпълнетието на дефинираните действия, в зависимост от изследваната входна комбинация.

6. План за тестване и тестови сценарии – предназначение и примерна структура

- Изготвя се от Ръководителя по качеството в проекта, съгласува се Ръководителя на проекта. По-скоро има характер на вътрешен документ, но в някои случаи може да се предостави на клиента по негово искане.
- Базира се на функционални и дизайн спецификации, както и на изисквания, договорени с процедурата за управление на промените в проекта;
- Съгласуван е с Плана на проекта;
- Създава се паралелно с техническата документация към проекта – т.е в началото на фазата за кодиране и тестване /development/
- Документ, който се развива и отразява промените в проекта;

6.1.Структура на плана

- Цел и предназначение на плана
- Обхват на тестовите активности
- Какво ще бъде тествано
- Какво няма да бъде тествано
- Референции към документи, на които се базира плана
- Критерии за достигане на планираното и договореното качество и за приемане на софтуера
- Описание на тестовия хардуер – тестов сървър(и), клиент, комуникации ..
- Описание на необходимия тестов софтуер
- Описание на необходими предпоставки и зависимости за провеждане на тестовете – наличие на набор с тестови данни,предварително обучение, /рискове?/ …;
- Инструменти за тестване, които ще се използват – за управление на грешките /Bug Tracking/, за автоматизирано тестване /Code Coverage, Memory Leaks, Functional, Stress, Performance …/ , Installation-monitoring tools, Virtual Machines
- Тестов екип – роли и отговорности, необходима подготовка, управление на тима (планиране на срещите на тима, срещи с клиенти …)

6.2. Примерен списък на групите тестове, които ще се извършат

- Тестове за качество на кода – code coverage, memory leaks …;

16. Управление на качеството на софтуерни приложения. Тестване на софтуер. План за тестване и тестови сценарии.

- Тестове на бизнес процеси / функционални тестове / - Списък на процесите, които ще бъдат покрити с тестове, с референция към съответстващите им тестови сценарии. Описание за реализация на regression testing;
- Тестове за производителност – списък и описание на скиптовете и референции към тях;
- Тестове за сигурност и права на достъп– списък и описание на тестовите сценарии с референции към тях;
- Тестове за архивиране и възстановяване след авария - списък и описание на тестовите сценарии с референции към тях;
- Тестове за инсталация - описание;
- Тестове на съпровождащата документация – описание;
- Тестове за приемане на системата – опционални, често се представят като отделна процедура за приемане (**User Acceptance Procedure**);

6.3.Тестови сценарии

Приоритет на тестовия сценарий – определя се от :

- Дали представя основен процес;
- Брой на клиентите, които работят с него;
- Дали е критичен за работата на останалата част на системата.

Тест сценариите идентифицират “дълбочината” на тестовия процес –респективно на степента на качеството.
Усилията за тестване на даден продукт могат да се оценят и на базата на множеството от тест сценарии – идентификация, разработка и изпълнение и отчетите от тях.
Тест сценариите се категоризират в зависимост от задачите, които изследват – функционалност, бързодействие, сигурност ...