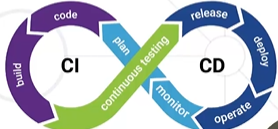
**QA BASICS**

* **Какво е Software Quality Assurance(SQA)?**
* Целта е да гарантира че софтуерът се държи според очакванията
* Методология за проверка на софтуера спрямо изискванията
* Работата на QA e софтуерно тестване – ръчно или автоматизирано
* Докладване на бъгове (дефекти) чрез системи за проследяване (bug tracking systems)
* Процесът се изпълнява от QA инженери

**1. Софтуерно тестване**

* Ръчно (manually click, check, confirm results)
* Автоматизирано (QA automation via scripts)
* Хибридно

**Непрекъсната интеграция и непрекъснато внедряване (CI/CD pipeline | Continues integration and Continues delivery)**

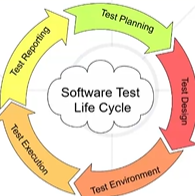
* Известяване (notification/report)
* Автоматизирано изграждане и обновяване в тестова среда (build and deploy)

**2. Софтуерното тестване е начин:**

* За оценка на качеството на софтуера
* Да се провери дали софтуера отговаря на определени изисквания и да се открият бъгове
* Да се намали риска от повреда на софтуера при неговото използване

**3. Процесът по анализиране на софтуерен продукт включва:**

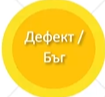
* Откриване на разликите между разработения софтуер и разписаните спецификации
* Оценка на функционалностите на софтуерния продукт
* **Основни цели на тестването:**
* Предотвратяването на дефекти (бъгове)
* Верификация на посочените изисквания
* Верификация на очакваното поведение на софтуера
* Да се намали нивото на риск от възможен софтуерен провал
* Да предоставя информация на заинтересованите страни (stakeholders/investors)
* Да спомага за спазването на договорни, законови или регулаторни изисквания



**Процесът по софтуерното тестване включва:**

* Планиране на тестването (какво, кога, как)
* Дизайн на тестването (test scenarios and test cases)
* Настройка на тестовата среда (инсталиране, конфигуриране, подготовка на тестови данни…)
* Реализация на тестовете (изпълнение на тестовете)
* Отчет на тестването (test reporting and bug find report)
* **Софтуерни дефекти (грешки, дефекти, бъгове и неизправности)**
* Човешка грешка/пропуск (human error)
* Пропуските водят до дефекти (бъгове в програмния код или грешки в изисквания/дизайна/друго)
* Ако дефектът бъде активиран това води до неизправност (функцията не изпълнява това което се очаква/изпълнява грешни неща)
* QA има за цел да намери дефектите
* Автоматизираното тестване и непрекъснатата интеграция/внедряване намаляват дефектите



**🡪 🡪**

* **Какво може да доведе до дефекти/бъгове?**

**1. Причини за грешки на програмиста/тестващия могат да бъдат:**

* Липса на време
* Недостатъчно добро обучение
* Сложен код
* Сложна инфраструктура
* Променящите се технологии

**2. Организационни фактори:**

* Неефективна комуникация (недоразумение)
* Неясно дефинирани изисквания

**3. Условия на околната среда (environment):**

* Електронни полета, магнетизъм, радиация, замърсяване и др. (могат да повлияят на състоянието на хардуера)
* Неправилна софтуерна среда (напр. грешен IP адрес)

**4. Други причини:**

* Неправилна конфигурация или неизправност в производствената и тестовата среда
* Некоректни тестови данни - (правилен тест който дава отрицателен резултат или грешен тест който дава положителен резултат)
* Некачествени тестове
* Невалидни очаквани резултати
* **Ръчно или Автоматизирано тестване**

**1. Ръчно тестване:**

* Тип софтуерно тестване което се изпълнява ръчно без използване на автоматизирани инструменти
* Човек изпълнява тестовете стъпка по стъпка без тест скриптове
* Тестовете се изпълняват индивидуално един по един

**2. Автоматизирано тестване:**

* Тип софтуерно тестване което се изпълнява автоматично чрез структура за автоматизация на тестове (automation framework)
* Тестерите използват инструменти и скриптове за да автоматизират повтарящи се дейности
* Включват писане на код(скрипт) и поддръжка на тестовете



* **Седемте принципа на тестването**

**1. Софтуерното тестване може да покаже наличието на дефекти, но не и отсъствието им**

* Тестването може да покаже наличието на бъгове
* Не може да докаже липсата на дефекти
* Подходящото тестване намаля вероятността за наличие на дефекти

**2. Изчерпателното тестване е невъзможно**

* Комбинациите от входни данни и тестови условия са безкрайни
* Да се тества всичко е непостижимо
* След направена оценка на риска, приоритет имат тестовете с най-висок за системата риск

**3. Ранното тестване спестява време и пари**

* Дейностите по тестването трябва да започнат възможно най-рано
* Трябва да са фокусирани върху предварително определени цели
* Колкото по-късно се открие един бъг – толкова по-висока е цената

**4. Струпване на дефекти**

* Тестването трябва да бъде подходящо насочено
* 80% от проблемите са породени от 20% от модулите в системата
* Фокусът пада върху 20% от които идват повечето проблеми

**5. Парадокс на пестицидите**

* Повтарянето на едни и същи тестове води до намаляване на ефективността им
* Неоткритите по-рано дефекти остават неоткрити

**6. Тестването зависи от контекста**

* Тестовете трябва да са съобразени и подбрани в зависимост от приложението за което ще се използват
* Софтуер изискващ високо ниво на безопасност се тества по различен начин от този за електронната търговия

**7. Заблудата „ Липса на дефекти“**

* Схващането, че софтуер с малък брой дефекти е успешен продукт е погрешно
* Самото намиране и отстраняване на дефекти е безсмислено ако изградената система е неизползваема или не отговаря на нуждите и очакванията на потребителите
* **Тест сценарии (test scenarios)**

**1. Тест сценарий**

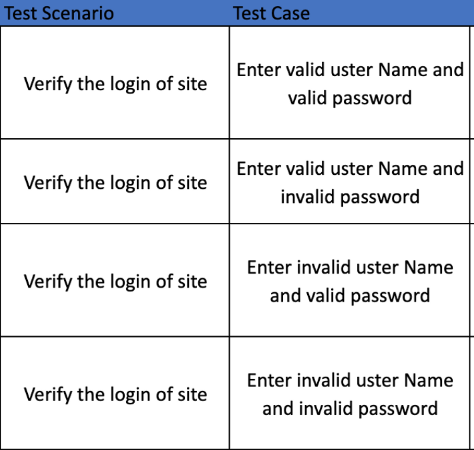
* всяка функционалност / свойство / потребителска история, която може да бъде тествана
* нарича се още “story under test” или “ feature under test” (напр. тестване форма за логин)

**2. Защо ни е необходим?**

* Сложните системи могат да бъдат разделени на няколко тест сценария
* Задава посоката в която ще се тества
* За изучаване на функционалността на програмата от край до край (end-to-end functioning)

**3. Един тест сценарий обикновено включва няколко тест случая**

**TS/TC for login example:**



**4. Писането на тест сценарии**

* Запознаване с документите и изискванията
* Как един потребител би използвал продукта (действия)
* Пишем тест сценарий за всяка функционалност (създаваме тестове който покриват очакваното и неочакваното потребителско поведение)
* Уверяваме се, че сме покрили всички изисквания
* Предават се сценариите за преглед
* **Тест случай (test cases) – тестове на единична конкретна функция**

**1. Какво представляват test cases:**

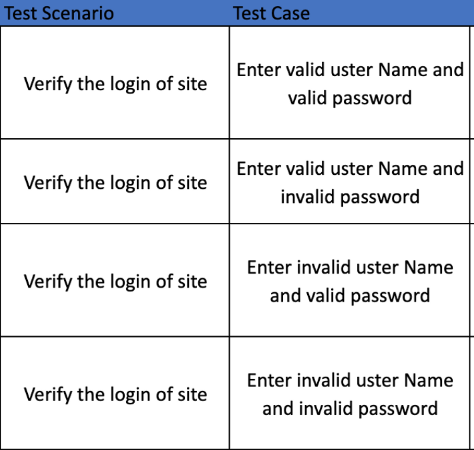
* Поредица от действия/стъпки чиято цел е да проверят конкретна характеристика или функционалност
* Поне два теста са необходими за тестване на определен сценарий (положителен и отрицателен тест)
* Включва специфични входни и изходни условия

**2. Защо са необходими?**

* За сравнение на очакваните с действителните резултати
* За проучване на функциониране на даден софтуерен компонент с определен вход и при определени входни условия
  + (пример за вход – поле за въвеждане на цифри, входни условия – в полето не може да се въвеждат букви, а само цифри)

**3. Тест случаите се състоят от:**

* Заглавие
* Стъпки за изпълнение
* Очакван резултат



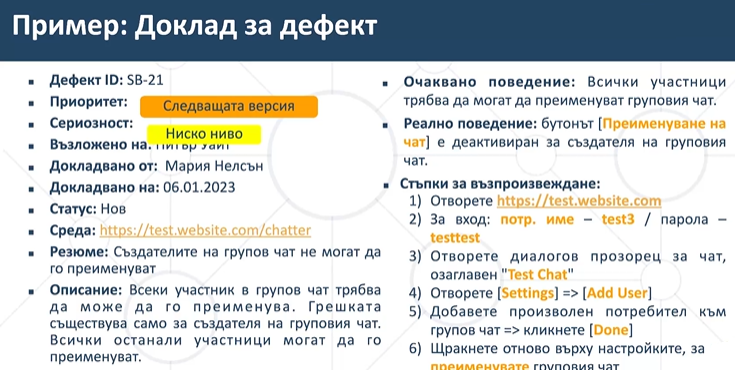
* **Доклад за дефекти (Bug report)**

**1. Какво представлява Bug report?**

* Писмен документ който описва определен бъг, открит по време на конкретна фаза в процеса на тестване

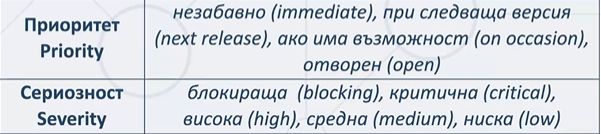
**2. Защо ни е необходим?**

* Предоставя подробна информация за проблема
* Помага при поддържането на архив за бъдещи справки
* Категоризира бъговете, за да помогне при анализ на първопричината
* Избягва се докладването на дублиращи се проблеми
* **Пример за bug report:**



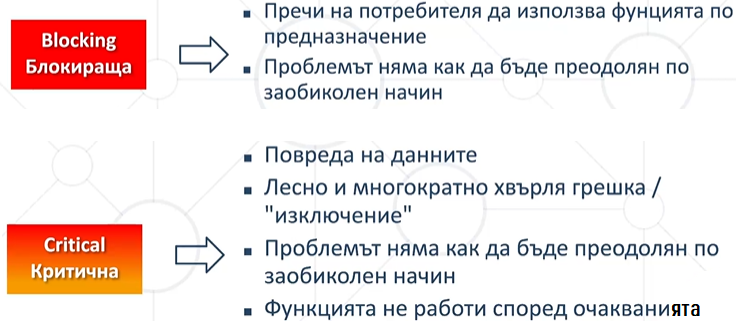
**3. Какво включва един Bug report**

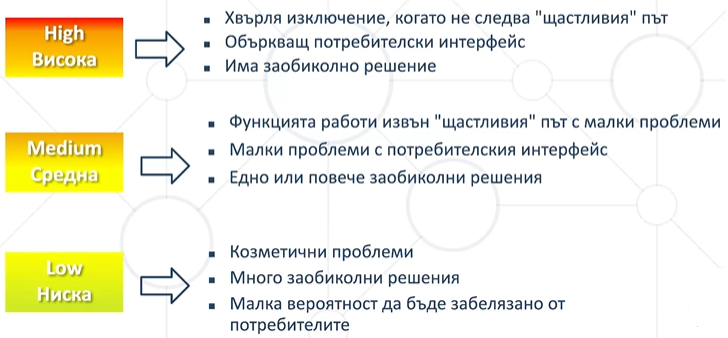
* Резюме – кратко заглавие на проблема
* Описание – включва аномалии, входни данни, очаквани и реални резултати
* Среда
* URLs
* Стъпки за възпроизвеждане
* Очаквано поведение (какво трябва се случи)
* Реално поведение (какво действително се случва)
* Препратки към външни източници, прикачени файлове (видеоклипове, снимки
* Всякаква допълнителна информация конфигурация
* Сериозност (severity) и приоритет (priority) – определят се от тестерите или на срещи (bug review/triage)
* **Примери severity/priority**



**4. Bug severity**

* Степента на въздействие, което даден бъг оказва върху работата на продукта
* Отнася се до функционалност или стандарти
* Индикатор за значимост на дефекта
* Определя се от функционалността
* Сериозността на дефекта е обективен показател и е малко вероятно неговият статус да се промени
* Базира се на техническата страна на продукта



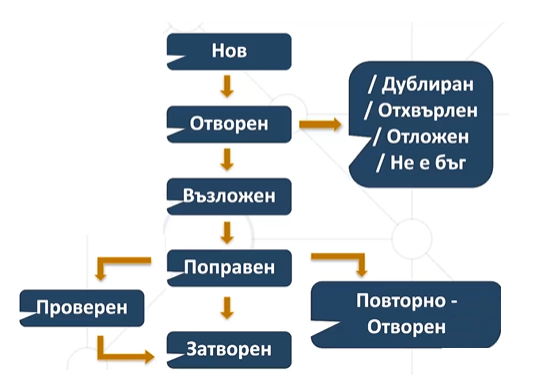


**5. Bug Priority**

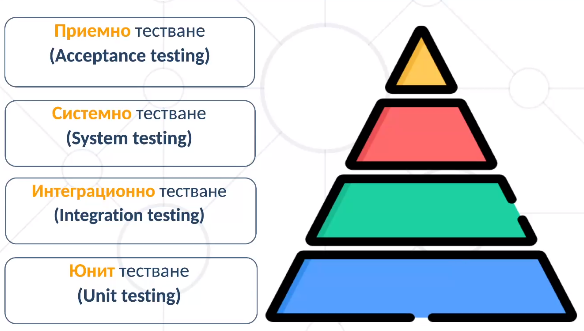
* Показва колко бързо трябва да бъде коригиран дефектът
* Определя реда в който бъговете трябва да бъдат отстранени
* Обвързан е с планиране / график
* Определя се след обсъждане с мениджъра / клиента
* В голяма степен зависи от финансови показатели
* Въз основа на изискванията на клиента
* Субективен показател и може да бъде променян във времето, според текущата ситуация по проекта
* **Примерна класификация**
* Незабавно / Висок – трябва да се отстрани възможно най-скоро
* Следваща версия / Среден – трябва да разреши в нормалния ход на дейностите по разработката. Може да изчака до следващ ъпдейт
* Когато има възможност / Нисък – може да бъде отложено, докато не бъде отстранена по-сериозна грешка
* Отворено – за сега не е планирано

**6. Жизнен цикъл на дефекта (Bug lifecycle)**

* Докладите за дефекти се управляват чрез жизнен цикъл
* Целта е процесът по коригиране на дефекти да се систематизира



* **Нива на тестване**
* Тестване за одобрение / приемно тестване (acceptance testing)
* Системно тестване (System testing)
* Интеграционно тестване (Integration testing)
* Юнит / Компонентно тестване (Unit testing)
* **Йерархия на тестовите нива**



**1. Unit testing – компонентно тестване**

* Първо и най-базово ниво на тестване, което се извършва преди интеграцията
* Необходимо е за да се провери дали отделните компоненти работят коректно
* Тества отделни компоненти на софтуера (компонент може да бъде отделна функция, метод, процедура, модул или обект)
* Обикновено се извършва от самите програмисти във фазата на писане на код
* Изпълнява се изолирано
* Позволява дефектите да бъдат отстранени рано, още във фазата на разработка

**2. Интеграционно тестване (integration testing)**

* Второ ниво от процеса по тестване на софтуер
* Извършва се от разработчици, тестери или интеграционен екип
* Отделните компоненти на софтуера се тестват в група
* Предполага се, че компонентите вече са тествани поотделно
* Тестването трябва да потвърди че всички свързани компоненти си взаимодействат правилно
* Основната цел е да се открият грешките в: интерфейс, взаимодействие между интегрирани компоненти и между системи
* **Поднива на интеграционно тестване**

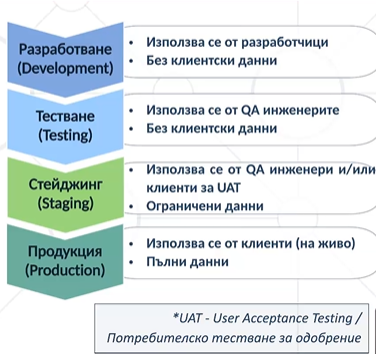
**2.1. Вътрешно интеграционно тестване**

* Разкрива дефекти в интерфейсите и взаимодействието между интегрираните компоненти
* “Integration test in the small”

**2.2. Външно интеграционно тестване**

* Тестване на системи взаимодействащи със други системи
* Тестване на интерфейси към външни организации
* “Integration test in the large”

**3. Системно тестване (system testing)**

* Трето ниво от процеса на тестване на софтуер
* Фокусира се върху цялата система
* Поведението на цялата система (какво прави системата)
* Възможностите на системата (как се справя системата)
* Реализира се чрез тестване от край до край (end-to-end,E2E)
* Извършва се от QA инженер
* **Пример за системно тестване**

**4. Acceptance testing (приемно тестване)**

* Последно ниво(4) обикновено преди внедряване (deployment)
* Валидира цялостното функционално бизнес решение
* Под внимание се вземат законовите/регулаторните изисквания
* Изпълнява се от членове на бизнес екипа (alpha testing) и от крайни потребители (beta testing)

**4.1 Защо е необходимо тестване за приемане от крайния потребител**

* Проверява работата на системата, обикновено преди внедряване
* Основната цел е работещо бизнес решение
* Не се фокусира върху козметични грешки
* Отговаря на въпроса дали актуалното поведение на системата съответства на очакванията на клиента
* **Типове тестване**

Група от тест дейности, който тестват специфични характеристики на определена софтуерна система. Разделят се на функционално и нефункционално тестване

**1. Функционално тестване**

* Отговаря на въпроса „Какво?“
* Тества функциите, които една система трябва да изпълнява (какво трябва да прави системата)
* Потвърждава дали софтуерната система отговаря на функционалните изисквания
* Основно включва тестване тип Black Box
  1. **Цели на функционалното тестване**
* Тестване на основните функции на приложение
* Съобщение за грешка (проверка дали излизат подходящи съобщения за грешка)
* Базисно приложение (безпроблемна навигация през различните екрани)
* Достъп (проверка на достъпа на потребителя до системата)

**2. Нефункционално тестване**

* Оценява надеждност
* Ефективност на работа
* Сигурност/безопасност
* Тества как и какво е качеството с което системата изпълнява функциите

**2.1. Цели на нефункционалното тестване**

* Фокусира се главно върху подобряване на качеството
* Лесно за употреба
* Ефективност
* Поддръжка
* Преносимостта на продукта

|  |  |
| --- | --- |
| Функционално тестване | Нефункционално тестване |
| Той тества функционалността на софтуера. | Той тества ефективността на функционалността на софтуера. |
| Той тества „Какво“ прави продуктът. Той проверява операциите и действията на дадено приложение. | Той проверява поведението на приложение. |
| Функционалното тестване се извършва въз основа на бизнес изискванията. | Нефункционалното тестване се извършва въз основа на очакванията на клиента и изискванията за производителност. |
| Той тества дали действителният резултат работи в съответствие с очаквания резултат. | Той проверява времето за реакция и скоростта на софтуера при определени условия. |
| Извършва се ръчно. Пример: Метод за тестване на черна кутия. | По-възможно е да тествате с помощта на автоматизирани инструменти. Пример: Loadrunner. |
| Той тества според изискванията на клиента. | Той тества според очакванията на клиентите. |
| Отзивите на клиентите помагат за намаляване на рисковите фактори на продукта. | Отзивите на клиентите са по-ценни за нефункционалното тестване, тъй като помагат за подобряване и позволяват на тестера да знае очакванията на клиента. |
| Функционалното тестване има следните видове: • Единично тестване • Интеграционно тестване • Тестване на системата • Изпитване за приемане | Нефункционалното тестване включва: • Тестване на производителността • Тестване на товара • Стрес тестване • Тестване на обема • Тестване на сигурността • Тестване на инсталацията • Тестване за възстановяване |
| Пример: Страницата за вход трябва да показва текстови полета, за да Въведете потребителското име и паролата. | Пример: Тествайте дали страницата за вход се зарежда за 5 секунди. |