12.Откриване и поправяне на дефекти. Тестване и настройване.

Качествен е този програмен продукт, който удовлетворява формулираните към него изисквания.

Всяко отклонение от изискванията ще наричаме дефект.

Под грешка ще разбираме неправилност, отклонение или неволно преиначаване на обект или процес.

Грешките могат да бъдат първични (неправилности в текста на програмите) или вторични (изкривявания на получените резултати ).

Съгласно друга класификация грешките могат да бъдат:

а)**технологични грешки** — при въвеждане на програмите или при подготовка на входните данни върху технически носители;

б)**алгоритмични грешки**;

в)**програмни грешки** — неправилно използване на конструкциите от съответните езици за програмиране;

г)**системни** — свързани с функциониране в определена операционна система.

Основни дейности за откриване и отстраняване на грешни са настройване (debugging) (локализиране и отстраняване на установени грешки) и тестване( (testing) изследване на програмите за установяване на съответствието им с различни по степен на формализираност характеристики, правила и изисквания).Когато настройването завърши, програмата решава някакъв проблем, а тестването доказва, че точно това е проблемът, който се иска да бъде решен.

Тестването се осъществява в три основни стъпки: планиране ( определя се целта на тестването, какво дасе тества, кога, с какви данни, как и кой да го извършва.), реализацията на тестването(описва се чрез сценарии за тестване, отчитането ( чрез анализ на документираните резултати и прилагане на критериите за изчерпателността и обхвата на тестване)

Има два основни подхода за тестване: структурен (да се изберат такива тестови данни, че да се осигури преминаване през всички програмни части на системата) и функционален (проверява се правилността на реализираните основни функции.В зависимост от избраните тестови данни и очаквани резултати тестването се дели на:детерминирано тестване - при зададени входни данни напълно е определено какви трябва да бъдат получените резултати; стохастично тестване — тестваните данни са случайни величини с определено разпределение и се знае разпределението на получаваните резултати.

В зависимост от начина на осъществяване тестването може да е низходящо, възходящо и смесено — започва от някакво междинно ниво и се провежда в двете посоки.

В зависимост от целта тестването може да бъде:

* за доказване на експлоатационната годност
* за атестация
* за пълна функционална проверка
* за проверка на специални програмни свойства
* за проверка на нови свойства или функции
* за проверка на работоспособността на системата в реални потребителски условия
* В зависимост от това, кой извършва тестването, то може да бъде:вътрешно тестване (от самите разработчици), независимо тестване (от експерти) и сертифициране( проверка и издаване на сертификат).

**V-модел за разработване и тестване на софтуер:** илюстрира връзката между фазите от класическия каскаден модел и съответните фази на тестване. Всяка фаза на разработване се свързва с фаза на тестване.

Преимущества:

* Фазите на тестване са представени на ниво, еднакво с това на фазите на разработване
* Резултатите от всяка от фазите на разработване могат да се проверят от групата по тестване, за да се осигури тяхната тестируемост достатъчно рано
* Своевременното планиране и проектиране на тестовете дава възможност за допълнителни проверки върху междинните продукти на фазите на разработване

**Автоматизиране на дейностите настройване и тестване:**Чрез автоматизация на настройването и тестването се постига:

* систематичност;
* подобряване организацията на тестване;
* повишаване надежността на програмната система;
* документиране на извършваните дейности;
* автоматично измерване обхвата на тестването.

Инструменталните средства, подпомагащи настройването, се наричат програми — дебъгери.

Основните възможности на дебъгерите са следните:

а) разглеждане на текста на програмата

б) трасиране на изпълнението

в) възможност за дефиниране на контролни точки

В зависимост от предназначението си програмните средства за тестване могат да бъдат:

а) **Анализатори на програми**

Статичният анализ е изследване на текста на програмите и извличане  
на определена информация от него. Статичният анализ може да бъде елемен-  
тарен или потоков. При елементарния статичен анализ текстът на програмата се проследява, като се създава таблица на използваните имена и срещанията им. Създава се статичен профил на програмата.

При потоковия анализ се построява управляващ граф на програмата и  
граф на обръщенията, отразяващ връзките между отделните програмни части.  
Потоковият анализ дава възможност да се открият няколко вида грешки:

-структурни — недостижими програмни части

-в дефинирането и използването на променливи

-несъответствия и грешки при предаване на параметри между различни програмни части и използването на тези програмни части и параметри.

Обикновено потоковият анализ се извършва на две нива - локално и глобално.

Потоковият анализ има два недостатъка:

* дълбочината на анализа се постига с цената на големи изчислителни ресурси
* поради наличието на условни или изчислителни преходи откриваните аномалии се делят на установени и на предполагаеми

Основни тенденции при статичните анализатори са разработването им за програми, написани на различни езици за програмиране.

При динамичния анализ програмата се изпълнява по управляем и систематичен начин и се изследва нейното поведение, за да се потвърди функционалната й коректност или некоректност.

При динамичния анализ се избира стратегия, подготвят с съответстващи на стратегията тестови данни и се определя начинът на тълкува не на получаваните резултати.

**Подход на черната кутия** - изследваната програмна час се разглежда като елемент с неизвестно съдържание. Към нея се подават входни данни и след изпълнението се получават определени резултати.   
„**Бяла кутия"** — програмата се инструментира по такъв начин, че да се натрупва информация за поведението й по време на изпълнение.

При метода на пробите за всеки изпълним оператор се дава броят на  
изпълненията му при конкретни входни данни. Показва не сам каква част от програмата е тествана, но и кои са най-често използваните програмни части. След изпълнението й да се създаде анотиран листинг, в който за всеки изпълним оператор броя на изпълненията му се дава и допълнителна информация.

Друг подход е вграждане в програмата на твърдения, които първоначално са като коментари, но могат да се активизират за целите на динамичните анализатори.

Формално-функционалните анализатори реализират „символично изпълнение на програмата". Програмата се разглежда като изчислима функция, която може да се декомпозира на частични функции.

Мутационните анализатори генерират мутанти на изследваните програми чрез внасяне на различни по вид и сложност грешки.

б) **Генератори на тестови данни** -тези програмни средства подпомагат или реализират съставянето на тестови данни в съответствие с избрана стратегия на тестване.

**в) Помощни програмни средства** - те подпомагат определени стъпки от процеса на тестване

г) **Интегрирани системи за тестване**- могат да подпомагат няколко метода на тестване или да реализират изцяло избран метод на тестване, като автоматизират планирането, генерирането на тестови данни, управляемото изпълнение, анализа на получаваните резултати и оценяване на ефективността на тестването .

**Осъществяване на тестването**

E=ket,дължината на времевия интервал t, E – усилия за отстраняване на грешки

**Метрики за тестване**: за обхвата на тестване – изпълнени инструкции, брой на тестовете, брой на тестваните пътища;за ефективност на тестването – открити дефекти, неоткрити дефекти, бизнес-полза, re-run загуби, проверени изисквания, предотвратени загуби; за ресурси за тестване – отностителна цена на тестването, средна цена на открит дефект, изпълнение на бюджета