# Проектиране и Тестиране на Софтуер
# ТУ, кат. КС, летен семестър 2012

# Лекция 41

Тема:

# ЖЦПС
# Тестиране и Настройка

# Съдържание:

- Въведение
- Част 1
- Тестиране на ПС
- Част 2
- Настройка на ПС
- Анализ и верификация на ПС

# Въведение

Дейностите тестиране и настройка са свързани с откриване, локализиране и отстраняване на грешки при реализация и изпълнение на ПС.

В лекциите за ЖЦПС бяха дадени определения за грешка и надеждност на ПО.

Когато се говори за тестиране и настройка, тематиката не е еднозначна. ПС се различават по сложност – обем, структура, период за разработка, брой програмисти в проекта. Естествено е при голямо разнообразие от ПС, представите за надеждност и грешки да се различават. Тук се излагат общи принципи, независимо от разнородността на ПС.

## Част 1

# Тестиране на ПС

# Съдържание:

- Тестиране на ПС
  - Определение
  - Видове тестиране

# Дефиниции за тестиране

1973, Bill Hetzel

Book: Program Test Methods

Definition: Testing is the process of establishing confidence that a program does what it is supposed to do.

# Дефиниции за тестиране

1979, Glenford Myers

Book: The Art of Software Testing

Definition: Testing is the process of executing a program or system with the intent of finding errors.

# 2 дефиниции за тестиране

*Определение 1:* Тестирането е процес на изпълнение на програма, потвърждаващ правилността на изпълнение на програмата и потвърждаващ, че в програмата няма грешки.

*Определение 2:* **Тестирането е процес на изпълнение на програма, с който се цели откриване на грешки в програмата**.

Първото определение е неправилно, тъй като то по същество е антоним на думата тестиране.

Коректно се счита второто определение.

# Дефиниции за тестиране

1983, Bill Hetzel

Book: The Complete Guide to Software Testing

Definition: Testing is any activity aimed at evaluating an attribute of a program or system. Testing is the measurement of software quality.

# Дефиниции за тестиране

200x, Computer Society:

Definition: Testing is a verification method that applies a controlled set of conditions and stimuli for the purpose of finding errors. This is the most desirable method of verifying the functional and performance requirements. Test results are documented proof that requirements were met and can be repeated.

# Определение

В потвърждение на правилната трактовка на тестиране е и известната мисъл на Дийкстра:

*Тестирането на програми може да служи като доказателство за наличие на грешки в програмите, но никога не може да докаже тяхното отсъствие.*

# Видове тестиране

- Source: R.a.Khan, pp 16
  - Black Box Testing
  - Unit Testing
  - Functional Testing
  - System Testing
  - Interoperability Testing
  - Volume Testing
  - Usability Testing
  - Recovery Testing
  - Object-Oriented Testing
  - Alpha Testing
  - Mutation Testing
  - Back-Out Testing
  - String Testing
  - White Box Testing
  - Integration Testing
  - Performance Testing
  - Regression Testing
  - Acceptance Testing
  - Stress Testing
  - Installation Testing
  - Security Testing
  - User Acceptance Testing
  - Beta Testing
  - Reliability Testing
  - Penetration Testing

# Видове тестиране

- Класификация на видовете тестове в зависимост от средата, в която се провежда тестирането на програмната система:
  - *Контролно тестиране (verification testing).*
  - *Изпитателно тестиране (validation testing).*
  - *Атестационно тестиране (certification testing).*

# Видове тестиране

Test-driven software development

Acceptance test, provided by customer

# Контролно тестиране

*Контролно тестиране (verification testing).* Опит да се открие грешка или грешки, като програмата се изпълнява в симулирана нереална среда в лабораторни условия с фиктивни недействителни данни. Този вид тестиране е познат и като алфа (α – alpha) тестиране.

# Изпитателно тестиране

*Изпитателно тестиране (validation testing).*
Опит да се открие грешка или грешки, като програмата се изпълнява в реална среда с достоверни данни. Този вид тестиране е известен и като бета (β – beta) тестиране.

Навярно са попадали бета версии на продукти. По този начин фирмата цели обратна връзка от потребителите за поведението на програмния продукт, който е в етап тестиране и настройка.

# Атестационно тестиране

*Атестационно тестиране (certification testing).* Това е авторитетна проверка за правилността на работа на една програмна система, като резултатите от изпълнението се сравняват с предварително известно множество от очаквани стойности.

# Видове тестиране

- Класификация на видовете тестове в зависимост от структурните елементи на програмната система, които се подлагат на тестиране:
  - *Автономно тестиране (module testing, unit testing).*
  - *Интегрално тестиране (integration testing).*
  - *Комплексно тестиране (system testing).*

# Автономно тестиране

*Автономно тестиране (module testing, unit testing).* Провежда се тестиране на отделен програмен модул в изолирана среда и независимо от всички останали модули. Автономните тестове имат отношение към вътрешната логика на модула и външната му спецификация за връзка с други модули.

# Интегрално тестиране

*Интегрално тестиране (integration testing).* Провежда се с цел контролиране на връзките между компонентите на програмната система. Този вид тестиране има отношение към структурата на отделната програма и архитектурата на цялата програмна система.

# Комплексно тестиране

*Комплексно тестиране (system testing).* Провежда се с цел контрол на програмната система по отношение на окончателните резултати, които се получават при изпълнение на програмната система.

# Аспекти на тестиране

- Освен основния аспект (ПРОВЕРКА ЗА ПРАВИЛНОСТ И/ИЛИ ОТКРИВАНЕ НА ГРЕШКИ), тестирането се разглежда и в следните аспекти:
  - Проверка за ефективност на реализацията – засяга се практикуваният стил на програмиране
  - Проверка за изчислителна сложност

# Видове грешки, откривани при тестиране

- Грешки при дефиниране на задачата
- Грешки при проектиране
- Грешки при програмиране
- Грешки при изпълнение

# Видове грешки, откривани при тестиране

- Грешки при дефиниране на задачата
  - Необходима е точна формулировка
  - Лошото поставяне на проблем води до програма, която решава правилно погрешна задача.
  - Опасност: необходимост от репрограмиране

# Видове грешки, откривани при тестиране

- Грешки при проектиране
  - Неправилно избран алгоритъм
    - Избран е **лош, неправилен** алгоритъм: числен метод за намиране корените на уравнение, но алгоритъмът се базира на разходящ итеративен процес
    - Избран е **слаб** алгоритъм: решава правилно, но работи бавно. Пример с възможни алгоритми за търсене в таблици

# Видове грешки, откривани при тестиране

- Грешки при програмиране
  - Синтактични грешки – не добро познаване синтаксиса на езика. Два вида грешки:
    - Предупреждения
    - Сериозни
  - Семантични грешки – не добро познаване смисъла на действието на отделен оператор от езика

# Видове грешки, откривани при тестиране

- Грешки при изпълнение
  - Логически грешки
    - Липса на начална стойност, смяна  клонове на условен оператор, неправилно формиран цикъл
  - Грешки от препълване
    - Изчерпване капацитета на таблици, буфери и тн
  - Грешки от некоректни данни
    - Важно: Валидация на входни данни
  - Грешки от хардуера

# Техники на тестиране

- Подобно на работата при етапи проектиране/програмиране на ЖЦПС, и тук при тестирането са възможни различни техники (стратегии):
  - *Тотално тестиране.*
  - *Възходящо (bottom-up) тестиране.*
  - *Низходящо (top-down) тестиране.*

# Тотално тестиране

*Тотално тестиране*. Програмната система се компилира и тества, като се проверява изпълнението за всяка от възможните комбинации входни данни. Този подход е възможен само в случаи, когато комбинациите входни данни са ограничен краен брой и времето за изпълнение е приемливо кратко.

# Възходящо тестиране

*Възходящо (bottom-up) тестиране*. ПС се компилира и тестира отдолу нагоре. Само модулите на най-ниско ниво се тестират автономно и изолирано. След тяхното тестиране, те се считат така надеждни, както функциите и методите от системните библиотеки. Следва тестиране на модули, които непосредствено викат вече проверените модули. Този път тестирането не е изолирано, а съвместно с модулите от по-ниските нива. Процесът продължава, докато се достигне до върха в йерархията на програмната система.

# Низходящо тестиране

*Низходящо (top-down) тестиране*. Програмната система се компилира и тестира отгоре надолу. Изолирано и автономно се тестира главният модул. След това към него се съединяват един след друг съставните модули, които главният непосредствено извиква. Получената комбинация се тестира. И тук процесът се повтаря, докато се комплектоват и проверят всички съставни модули. При низходящото тестиране е важно да се изяснят два въпроса:

# Низходящо тестиране

а/ Как се процедира, ако тестираният модул извиква модул, който още не е тестван? Липсващият модул се замества с фиктивен, който има входна точка и изход в извикващата среда. Това са празни програмни единици (stubs), които в най-прост случай имат следната структура:

**заглавие** – входна точка

**< празно тяло>**

**return** – логически край

**end** – физически край

# Низходящо тестиране

б/ КАК и В каква форма се подготвят тестовите данни и как се подават на програмата? Тестовите данни се подготвят така, както се готвят входни данни за всяко друго изпълнение. При добре проектираните програми входните операции са извън главната програма и са концентрирани в самостоятелни модули. Това води донякъде до отклонение от строгия низходящ принцип, но води до по-ефективно изпълнение на входно/изходните операции.

# Данни за тестове

Тестирането е свързано с подбирането на подходящи комбинации входни данни. Тази дейност е известна като *проектиране на тестове* и не е редно да се извършва от автора на програмата. В софтуерните фирми се формират специални групи за тестиране от високо квалифицирани специалисти (QA engineers). Microsoft: тестващата група през 1984 е наброявала 5 човека, а през 1993 е нарастнала на повече от 500.

# Данни за тестове

Изобщо висококачественото проектиране на тестове е сложен и отговорен процес. При него се изисква както творчество, така и известна доза разрушителна сила на духа.

Основен принцип при проектирането на тестове е да се подготвят тестови данни за всеки клон на алгоритъма, всяка комбинация допустими стойности от входни данни.

# Данни за тестове

При участък с разклонен алгоритъм е необходим тест, който ще гарантира изпълнение на всички условни преходи във всички разклонения.

При участък с цикъл е необходим тест, който ще изпълни цикъла 0 пъти, тест, който ще изпълни цикъла 1 път, тест, който ще изпълни цикъла максимален брой итерации.

# Данни за тестове

Въвежда се правилото на *минимален критерий* – най-малко по едно изпълнение за всички клонове на алгоритъма.

Подборът на данните за тестовите поредици трябва да обхваща следните различни случаи:

1. Работа с тестови данни – нормални случаи;
2. Работа с тестови данни – гранични случаи;
3. Работа с тестови данни – изключения;
4. Работа с тестови данни – *нулеви случаи*.

# Данни за тестове - *нулеви случаи*

За аритметичните данни това е стойност нула.

За символните данни това е низ от празни позиции – интервали, или низ с нулева дължина.

За указателите това е специалната стойност nil в езика Pascal и NULL в езиците C/C++.

Набор от входни данни, при които програмата не изпълнява никакви действия, се нарича *нулев вариант*.

# Пример за тестиране

- Да се тества програма, която брои редовете, думите и символите във входния поток (Unix *wc <input>*). Програмата извежда:AA BB CC
  - AA – брой редове
  - BB – брой думи
  - CC – брой символи
- Какви тестови набори са подходящи за откриване на бъгове, ако има?
- Преди да ги подберем, ще припомним:

# Пример за тестиране

- Характерни особености на тестирането
  - Доказва, че има грешка
  - При правилен тест – локализира грешката
- Ред на тестиране
  - Леки тестове:
    - С правилни входни данни
    - С по-малка (но не гранично малка) рамерност за лесна проверяемост
  - Тежки тестове:
    - За граничите случаи
    - За неправилни входни данни

# Пример за тестиране

| Тестов набор | Очакван резултат | | |
|---|---|---|---|
| | редове/думи/с-ли | | |
| Без данни | 0 | 0 | 0 |
| 1-символна дума | 1 | 1 | 2 |
| 2-символна дума | 1 | 1 | 3 |
| Две 1-символни думи в 1 ред | 1 | 2 | 4 |
| Две 1-символни думи в 2 реда | 2 | 2 | 4 |

# Пример за тестиране

Тестови набори. Гранични случаи:

- Без данни
- Без думи – само празни редове
- Без думи – само бели шпации, интервали
- 1 дума на ред, без интервали
- Дума в началото на ред
- Дума в края на ред, след интервали
- И т.н ...

# Правила за тестиране

- Поставяйте входни филтри
- Не превръщайте програмите в капан
- Осигурявайте циклите
- Използвайте готови, изпробвани решения
- Създайте своя помощна библиотека от функции, класове
- Не оставяйте нетествани клонове от програмата

# Аксиоми за тестиране

- Добър тест е този, който открива грешки, а не този, който демонстрира правилна работа.
- Проблем при тестването е да решим кога да свърши тестването.
- Невъзмжно е де се тества напълно собствената програма.
- Да се избягват невъзпроизводими тестове.
- Да се подготвят тестове за правилни и неправилни данни.
- Детайлно изучавайте резултата на всеки тест.
- Колкото повече грешки се откриват, толкова расте вероятността за наличие на неоткрити грешки.
- Тестването – дело на най-способни програмисти.
- Тестопригодността на ПС – ключова задача.

# V&V in Software Testing

## Verification

**Are we producing the product right?**

## &

## Validation

**Are we producing the right product?**

**- Boehm**

# V & V

**Verification** ensures the product is designed to deliver all functionality to the customer; it typically involves reviews and meetings to evaluate documents, plans, code, requirements and specifications;

**Validation** ensures that functionality, as defined in requirements, is the intended behavior of the product; validation typically involves actual testing and takes place after verifications are completed.

# V & V

**Verification** is set of activities to ensure that the function that is developing works properly

**Validation** is a different set of activities to ensure that the function that has been built is working properly.

# V & V

**Verification** is a process in which information is checked using accurate measures. E.g. when you enter a new password you are asked to retype it to verify that the password supplied is correct.

**Validation** however is the automatic process in which rules are applied in order to make information correct,e.g. if the right type of data is entered in a certain cell in a database.

# V & V

**Verification:**

1. **Verification is Correctness.**
2. **It is conducted by QA team.**

**Validation:**

1. **Validation is Truth.**
2. **It is conducted by development team with the help from QC team.**

# models to verify designs and validate requirements

- Verification and validation techniques applied throughout the development process enable you to find errors before they can derail your project.
- Most system design errors are introduced in the original specification, but aren't found until the test phase.
- When engineering teams use models to do virtual testing early in the project, they eliminate problems and reduce development time by as much as 50%.

# Част 2

# Настройка на ПС

# Съдържание:

- Определение
- Методи за настройка

# Тестиране и Настройка

С тестирането се свързва още една дейност – *настройка (debugging)* на ПС. Погрешна е представата, че настройката е разновидност на тестирането. Двете дейности не бива да се се смесват. Между тях има връзка и разлики:

- Ако една програма очевидно не работи правилно, тя се настройва.
- Ако една програма видимо работи правилно, тя се тества.

# Тестиране и Настройка

Тестирането е дейност, с която се открива наличие на грешки.

Настройката е дейност, с която се установява точната природа на грешката. Следва локализиране на грешката и нейното отстраняване.

Това са 2 свързани, но не идентични дейности.

Обикновено резултатите от тестирането служат като начални данни, с които стартира настройката.

# Типове настройка

Настройката започва след корекция на синтактичните грешки. Следва изпълнение с тривиални входни данни.

Ако се получат достоверни резултати, тестването продължава с друг набор входни данни.

Обратно, ако програмата не дава верни резултати, тя следва да се настрои с този набор данни, като са възможни следните случаи:

# Типове настройка

- Програмата се транслира, изпълнява, но не дава резултат.
- Програмата се транслира, но изпълнението завършва преждевременно.
- Програмата се транслира и изпълнява, но дава неправилен резултат.
- Програмата се транслира, но при изпълнение зацикля.

# Типове настройка

- Програмата се транслира, изпълнява, но не дава резултат.
  - Дължи се на логическа или системна грешка.
  - Логическа грешка – правят се проверки, които водят до изход без печат.
  - Системна грешка. Изпълнението се суспендира от хардуера, ОС, самата програма. Обикновено следва системно съобщение, което не винаги е информативно за потребителя. (Null pointer assignment)

# Типове настройка

- Примери с причини за системни грешки:
  - Деление на 0
  - Предаване на управлението в защитена област на паметта
  - Неправилни индекси на масив, грешна адресация
  - Препълване или загуба на стойност при аритметични операции

# Типове настройка

- Програмата се транслира, но изпълнението ѝ завършва преждевременно.
  - Програмата завършва не там, където очакваме. Оценката е, че е допусната груба грешка, в резултат на което по-нататъшно изпълнение е невъзможно.

# Типове настройка

- Програмата се транслира и изпълнява, но дава неправилен резултат.
  - Оценката е, че това е относително добър резултат, тъй като скелетно програмата е правилно структурирана.

# Типове настройка

- Програмата се транслира, но при изпълнение зацикля.
  - Този тип грешки се откриват лесно. Достатъчно е да се вмъкне контролен печат преди и след съмнителни цикли.
  - Препоръка: не вътре в цикъла, за да се избегне извод на екрана на безконечна поредица

# Подходи за настройка

В практиката са се наложили различни подходи, методи и средства за настройка (локализация на грешки) при изпълнение на една програмна система, които се свеждат до следното:

1. Метод на грубата сила;
2. Интелигентни методи.

# Метод на грубата сила

а/ Добавяне в програмата на контролен печат. Тази идея не е лишена от смисъл, но е пример за безсистемен подход Ако този принцип се възприеме, ето някои препоръки:

1. Извеждане на ехо печат на входни данни;
2. Извеждане на информация за развитието на числовите пресмятания и за логическата последователност при изпълнението на ПС;
3. Установяване на програмни флагове за включване на контролния печат в случай, че се работи с тестовите версии на ПС.

# Метод на грубата сила

б/ Извеждане съдържанието на определени области от паметта (дъмп на паметта).

в/ Стъпково проследяване изпълнението оператор по оператор.

г/ Установяване на условни и/или безусловни точки на прекъсване (break points) и автоматично изпълнение с прекъсване при достигане на съответна точка на прекъсване.

# Метод на грубата сила

Изброените дейности могат да се извършват ръчно и автоматизирано. В първия случай се изискват умствени усилия. Във втория случай се налага познаване на средства за настройка, които предлагат програмните среди като програмите Debug.exe, Symdeb.exe, Afd.exe, вграден дебъгер в интегрираните програмни среди и автономна програма Td.exe (Turbo Debugger) и други

# Интелигентни методи

Интелигентните методи се основават на систематично формулиране на ситуациите с изграждане на множество възможни хипотези и тяхната последваща проверка.

# Интелигентни методи

1. Изключване на всички части от програмата, които не са причина за грешка.
2. Генериране на нови хипотези  N. Ако не може, то груба сила.
3. Цикъл  for (I=1;I<=N;I++)
   {
   проверка на I-тата хипотеза
   Ако I-тата хипотеза е вярна, то Край, успех.
   }
4. Всички хипотези са изчерпани. Преход към 2 или груба сила

# Груба сила и/или Интелигентни методи

Практическата работа по тестиране и настройка на една програмна система най-често включва елементи и подходи както от грубата сила, така и от интелигентните методи.

# Правила/аксиоми за настройка

- Мисли!
- Споделяй с друг, ако си блокирал
- Отложи работата за утре, ако си уморен
- Не възлагай големи надежди на средствата за настройка
- Не експериметирай сляпо

# Принципи за локализация

- Където има 1 грешка, вероятно има и други грешки.
- Търсете грешки, а не симптоми.
- Не мислете, че сте откртили 100% грешки.
- Внимание! Опасност от внасяне на нови грешки при корекцията.
- Не внасяйте корекция, преди да сте усвоили напълно алгоритъма.
- Коригирайте първичен текст, а не обектен код

# Анализ и Верификация

# Анализ и верификация

Тестирането и настройката имат своя алтернатива и тя се нарича *анализ и верификация на ПС*. Невъзможността да се провежда тотално тестиране и трудностите с подбирането на подходящи тестови поредици са довели до следната алтернативна идея. Вместо тестиране на ПС, провежда се *доказателство, че програмите правилно се изпълняват*. Процесът на доказателство на правилността на програмното изпълнение се нарича още аналитична верификация. Основната идея на този подход е следната.

# Анализ и верификация

На всеки оператор от програмата се съпоставят два булеви израза – предикати. Единият изразява знанието за състоянието на програмата преди изпълнението на оператора. Другият изразява знанието за състоянието на програмата след изпълнението на оператора. Формално тези предикати се записват като коментари в програмните текстове:

C/C++

```
/* предусловие */
< оператор >
/* постусловие */
```

# Анализ и верификация

Задача: да се докаже за всеки оператор истинността на постусловието, ако предусловието е вярно. Разглеждат се оператор по оператор. Започва от първия и се търси доказателство, че от предусловието на първия оператор следва постусловието на последния оператор на програмата. Релацията между първо предусловие и последно постусловие характеризира спецификацията на програмата (какво тя прави). Тогава доказателството, че постусловието следва от предусловието, е равностойно на твърдението, че програмата ще се изпълнява в съответствие със своята спецификация.

# Благодаря
# За
# Вниманието