TEMA 5
QUALITAT DEL SOFTWARE

1. Introducció
2. Models de proves de software
3. Estratègies de prova
4. Els problemes del mal disseny de software
5. Principis bàsics del disseny de software OO
6. TDD: Test-Driven Development

5.1

Introducció

Introducció

- **Forat no detectat a la capa d'ozó**

  El forat a la capa d'ozó sobre l'Antàrtida no es va detectar durant un llarg període de temps (7 anys) pel fet que el programari d'anàlisi de dades utilitzat per la NASA en el seu projecte per mapejar la capa d'ozó havia estat dissenyat per ignorar els valors que es desviaven molt de les mesures esperades.

Introducció

- **Radioteràpia mortal**

  Es van administrar sobredosis massives de radiació a pacients entre 1985 i 1987. Diversos pacients van rebre fins a 100 vegades la dosi prevista, i almenys tres d'ells van morir com a conseqüència directa de la sobredosi de radiació. El motiu va ser que el dispositiu de radioteràpia (Therac-25) tenia defectes de concurrència en el programari de control.

Introducció

- **Errors en llançament de coets**

  Degut a un problema de software, al 1996 el coet europeu Ariane 5 es va destruir 37 segons després del llançament. El programari defectuós va fer desviar el coet de la seva ruta. El problema va ser la reutilització de codi del sistema de llançament del predecessor Ariane 4, que tenia condicions de vol diferents. Més de 370 milions de dòlars es van perdre a causa d'aquest error.

5.1

Introducció

- **Descobert al banc de Nova York**

  Al novembre de 1985, el Bank of New York (BoNY) va tenir accidentalment un descobert de 32 bilions de dòlars a causa d'un comptador de 16 bits que mai va ser verificat i que es va “desbordar”. Aquest error va provocar que el BoNY no pogués processar nous préstecs, i la Reserva Federal va haver de fer una transferència automàtica $24 bilions al banc per cobrir les seves despeses. El banc va haver de pagar $5 milions d’interessos diaris fins que el programari es va arreglar.

Introducció

- **Bugs d'aplicacions mòbils**
  - WhatsApp: un bug permetia als usuaris accedir a les dades d'altres comptes o espiar missatges
  - FaceBook: un bug permetia insertar imatges amb les que infectar-nos amb ransomware

5.1

Introducció

- És **impossible** desenvolupar software **sense errors**

- El **cost** de trobar errors en entorns de producció pot arribar a ser **molt gran**, econòmicament i/o en vides

Introducció

Què és un Test ?

Avaluació d'un sistema de forma manual o automàtica per tal de **verificar** que satisfà els **requisits especificats** o per identificar **diferències** entre els **resultats esperats** i els **obtinguts** (IEEE, 1983)

Introducció

Objectius dels Tests

Garantir que el **sistema** satisfà els **requisits** especificats

Garantir que el sistema proporciona els **resultats esperats**

Introducció

"Any program feature without an automated test simply doesn't exit"

Kent Beck

5.2

Models de prova de Software

Models de prova de Software

☛ La **majoria** dels **defectes** tenen la seva **causa** en una **mala definició** dels **requisits**.

El cost de corregir un error és més barat quan més aviat es troba

Models de prova de Software

Un **defecte** de **codificació** implica **recompilar**, però un **defecte d'arquitectura** o disseny implica **redissenyar i re-especificar,** a més de generar de nou els manuals

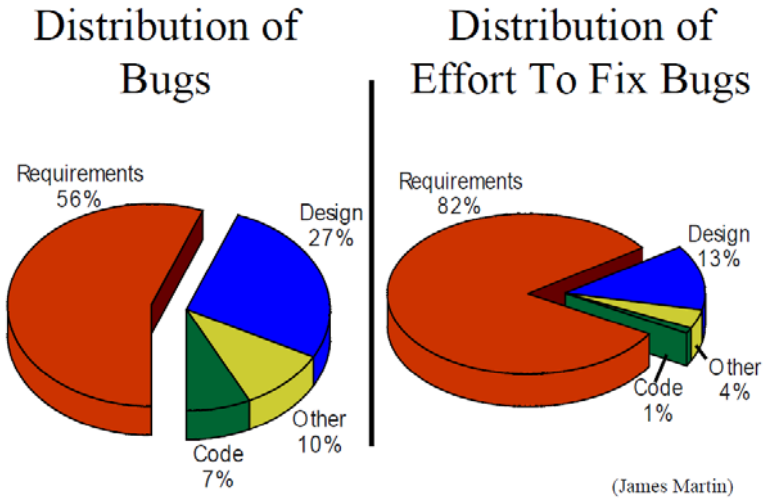


Figure 1: *Distribution of Defects – James Martin*

Models de prova de Software

Model seqüencial

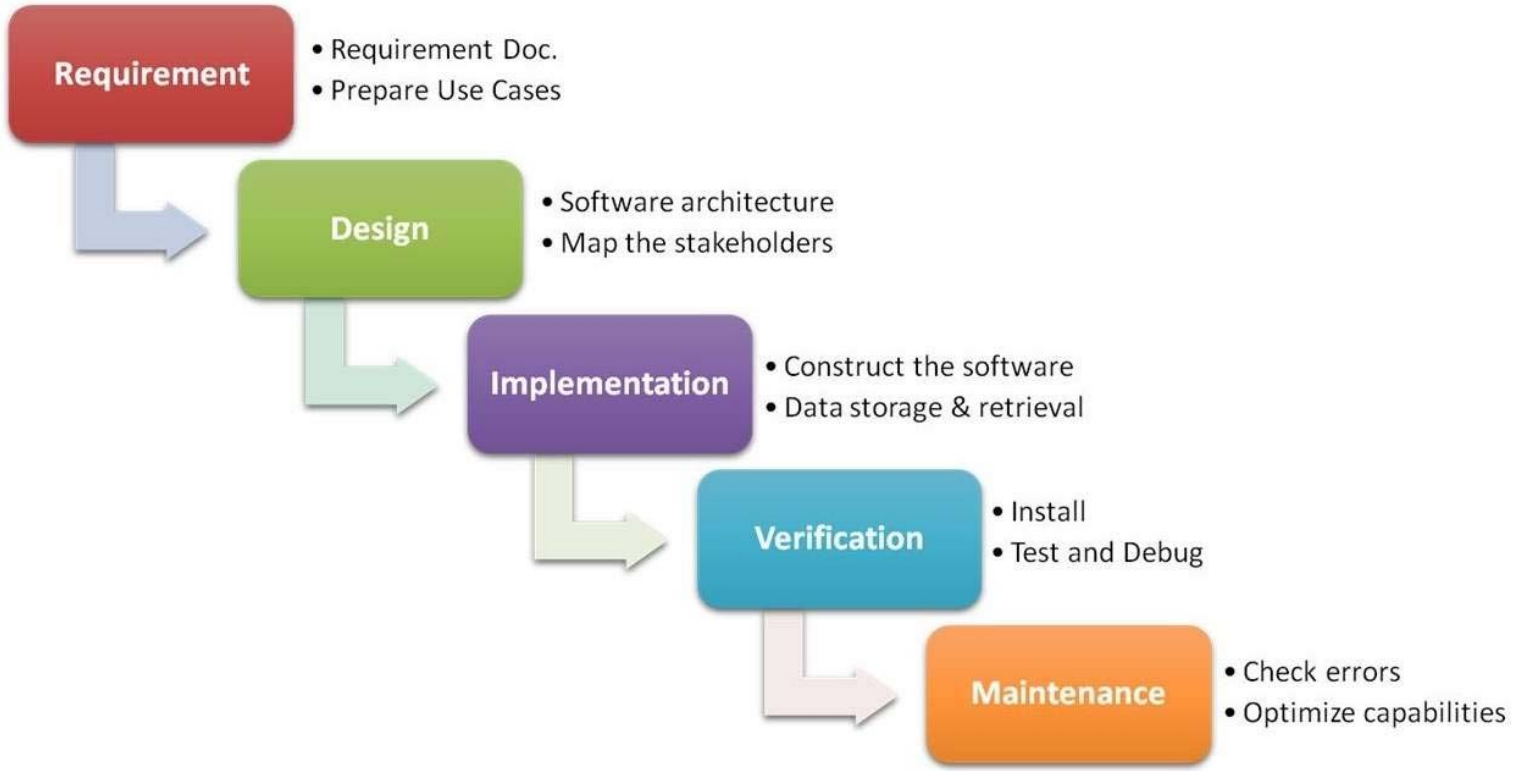
En un model seqüencial de desenvolupament de SW, cada fase està dissenyada per una activitat específica. La fase de test comença quan la fase de desenvolupament ha acabat.

El client veu el producte quan ja està acabat.

Els **errors** en els requisits, d'arquitectura i disseny es **detecten al final del projecte**.

Models de prova de Software

☛ Model de desenvolupament en Cascada



Models de prova de Software

Model en V

És un model incremental. Té **una fase de test associada** a cada **fase de desenvolupament**.

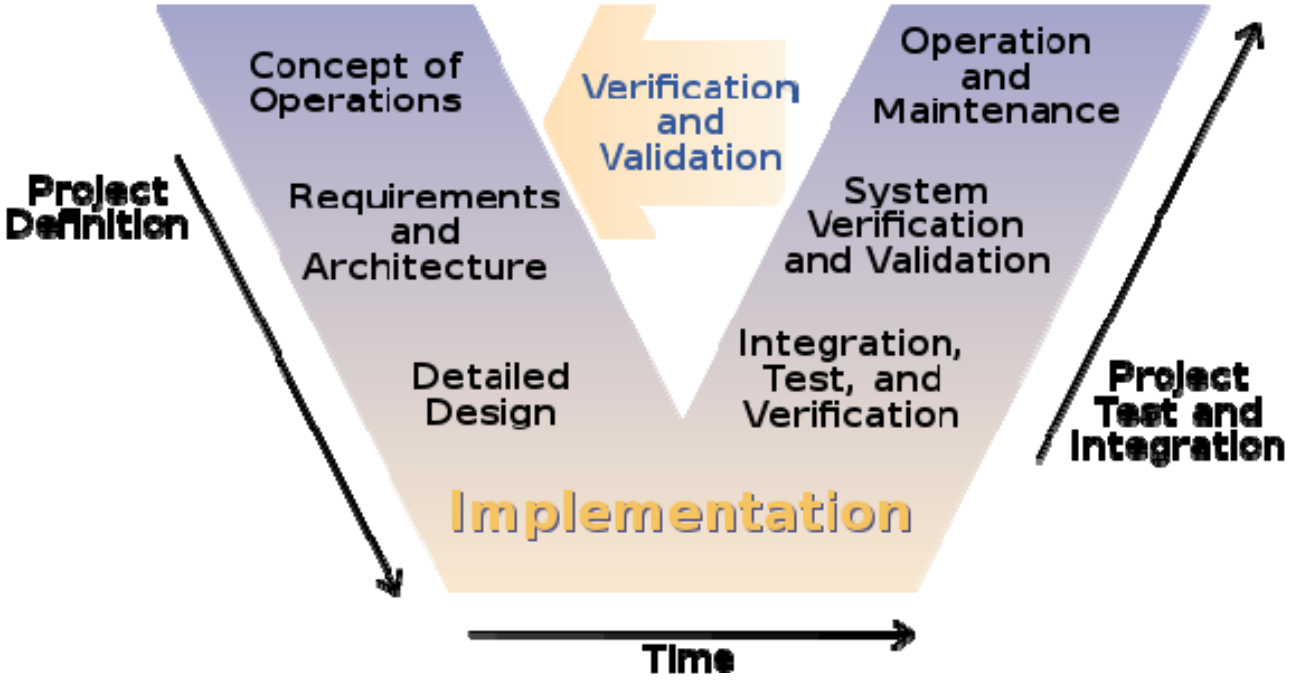
Els **errors d'arquitectura i disseny** es detecten en **fases inicials** del projecte.

El client veu el producte quan ja està acabat.

Els **errors** en els **requisits** es detecten al **final** del **projecte**.

Models de prova de Software

Model en V



Models de prova de Software

Model Agile

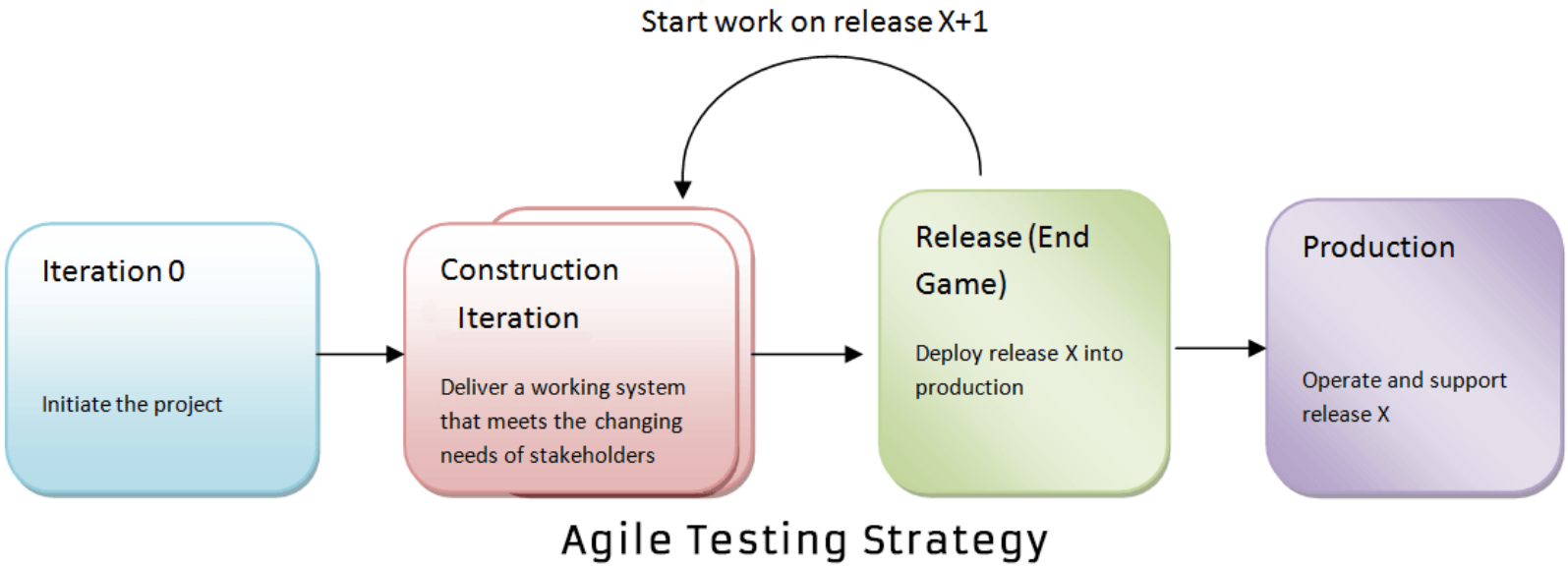
És un model **d'integració continua entre desenvolupament i test**

El **test comença** en les **fases inicials**. Permet aplicar de forma efectiva **TDD (Test Driven Development)**.

Demanda molta **implicació** del **client** per a que provi els prototips i doni feedback (versions *beta*).

Models de prova de Software

Model Agile



Agile Testing Strategy

Models de prova de Software

Model Agile

Els **errors** de **requisits, arquitectura i disseny** es detecten **abans** i el **cost** d'arreglar-los és molt **baix**.

El **sistema** és **estable** des del **principi**.

Model en V vs Model Agile

La **diferencia no** és en el **tipus** de tests que es fan, **sinó** en **com i quan es fan**.

El model **Agile** permet tenir **feedback del client** en **fases molt inicials** del projecte i permet adaptar-se més fàcilment als canvis en els requisits.

5.3

Estratègies de prova

Estratègies de prova

- El SW que va a producció requereix haver estat provat

- L'objectiu és entregar SW més ràpid sense sacrificar-ne la qualitat

- Existeixen diferents **tipus** de prova:
  - Caixa Negra / Caixa Blanca
  - Manuals / Automàtics

Estratègies de prova

Proves de Caixa Negra

Es realitzen quan no es coneix l'estructura interna del SW.

Estratègies de prova

Proves de Caixa Negra

Permeten provar el comportament del sistema i que es compleixen els requisits.

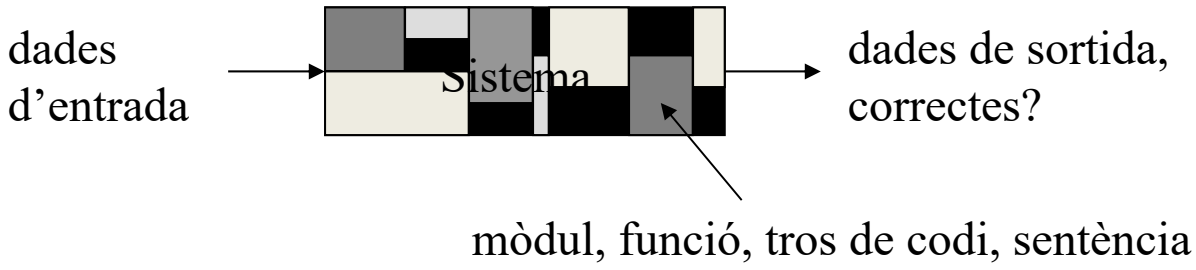
Objectius:

- Garantir que les **interfícies internes funcionen** correctament
- Garantir que l'accés a bases de dades i **serveis externs funcionen correctament**
- Garantir que no hi ha errors de rendiment, inicialització, finalització, etc.

Estratègies de prova

Proves de Caixa Blanca

Es realitzen quan es coneix l'estructura interna del SW.



Estratègies de prova

Proves de Caixa Blanca

Permeten realitzar un examen minuciós de la lògica interna d'un sistema.

Objectius:

- Garantir que **s'executen totes les línies de codi**, incloses les excepcions
- Garantir que **s'usen totes les estructures de dades**
- Portar fins al **límit l'execució** dels **bucles**

Estratègies de prova

Tests Manuals

Desenvolupar, provar i desplegar SW que creix constantment de manera manual i ràpidament és impossible.

Cal desplegar el SW en un entorn de proves i executar tests de caixa negra per comprovar si tot funciona correctament.

Consumeix molt temps, i és una tasca **repetitiva** i **avorrida** que **deriva en errors**.

Estratègies de prova

Tests Automàtics

Permeten saber, en segons, si el SW s'ha trencat.

Tenir un model de test de SW efectiu permet als equips desenvolupar SW més ràpid i amb més confiança.

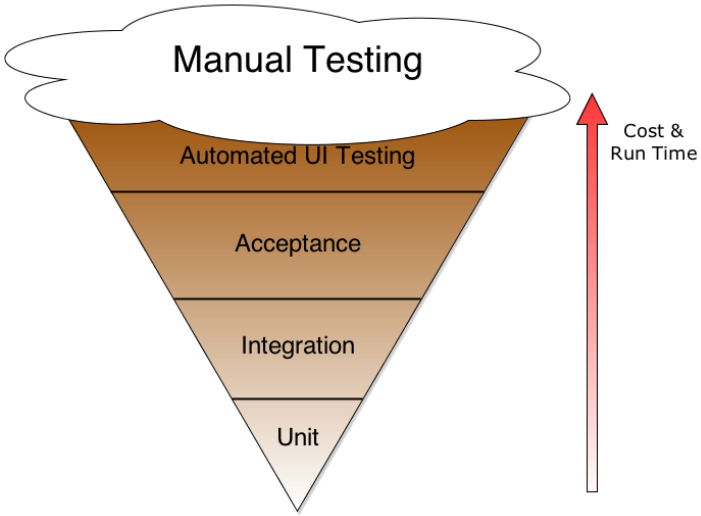
Com fer-ho?

Dividint els **tests** en **capes** i decidint quins i quants tests fer en cada capa.

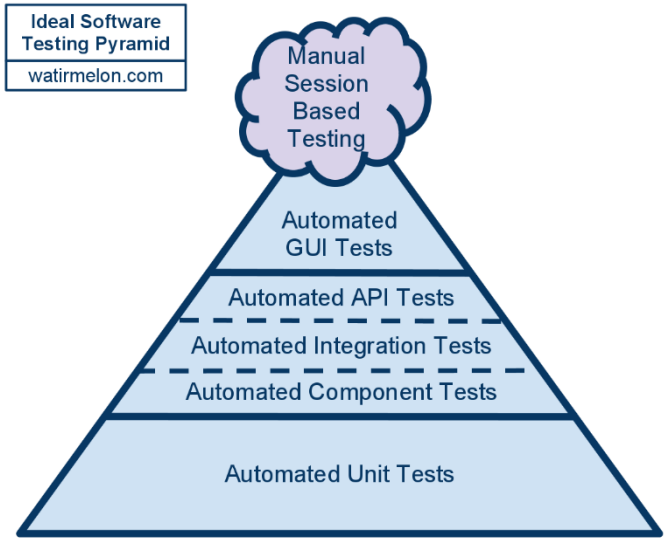
Estratègies de prova

“Test Pyramid” (M. Cohn)

Indica quins tipus de proves s’haurien de trobar en els diferents nivells de la piràmide



Tradicional



Agile

Estratègies de prova

Regles per definir els tests

Nivell més Baix (proves de Caixa Blanca/Negra):

- Més tests
- Més aillats
- Més ràpids

Nivell més alt (proves de Caixa Negra):

- Menys tests
- Més integrats
- Més lents

Estratègies de prova

Tests unitaris

Són proves de Caixa Blanca i Caixa Negra

S'inclouen en la capa més Baixa de la pirámide

Asseguren que una "*unitat*", o SUT ("*System Under Test*") del repositori funciona com s'espera

En un llenguatge OO, una "*unitat*" pot ser des d'una **funció** a una **classe**.

Estratègies de prova

Tests unitaris

Una classe de test per cada classe.

Una classe de test ha de provar, com a mínim, la interfície pública d'una classe.

Els tests unitaris han de provar:

- El happy-path (flux principal)
- Els non-trivial paths
- Els corner cases

Estratègies de prova

Codi de prova

```
public class MoneyTest extends TestCase
{
   Money f12CHF, f14CHF;

   protected void setUp()
   {
      f12CHF = new Money(12,”CHF”);
      f14CHF = new Money(14,”CHF”);
   }

   public void testSimpleAdd()
   {
      // [12 CHF] + [14 CHF] == [26 CHF]
      Money expected = new Money(26, "CHF");
      Money result   = f12CHF.add(f14CHF);

      assertTrue(result.amount()==expected.amount());
      assertTrue(result.currency()==expected.currency());

      assertEquals(result, expected);
   }

   public void testConstructor()
   {
      assertTrue(f12CHF.amount()==12);
      assertEquals(f12CHF.currency(),”CHF”);
   }

   public void testMoneyEquals()
   {
           Money equalMoney = new Money(12,"CHF");
           assertEquals(f12CHF, equalMoney);
           assertFalse(f12CHF.equals(f14CHF));
           assertTrue(!f12CHF.equals(null));
           assertEquals(f12CHF, f12CHF);  }

}
```

Codi desenvolupat

```
public class Money
{
   private int iAmount;
   private String sCurrency;

   public int amount(){ return iAmount; }
   public String currency(){ return sCurrency;}

   public Money(int amount, String currency)
   {
      iAmount = amount;
      sCurrency = currency;
   }

   public boolean equals(Object anObject)
   {
      if (anObject instanceof Money)
      {
            Money aMoney = (Money)anObject;
            return
(aMoney.currency().equals(currency())) &&
                  (amount() == aMoney.amount());
      }
      return false;
   }

  public Money add(Money m)
  {
   return new Money(amount()+m.amount(),currency());
  }

}
```

Estratègies de prova

Tests d'integració

Són proves de Caixa Negra/Blanca

Totes les aplicacions no trivials s'integren amb altres parts: **bases de dades**, **sistemes de fitxers**, **xarxes**, **altres aplicacions**, etc.

Els tests d'integració proven la **integració** de **l'aplicació amb totes les parts** amb les que interacciona.

Estratègies de prova

UI Tests

Són proves de Caixa Negra

Asseguren que la **interfície** de l'aplicació **funciona correctament**.

Poden provar comportament, layout, o usabilitat.

Estratègies de prova

Tests d'Acceptació (Tests Funcionals)

Són proves de Caixa Negra

Comproven si l'aplicació funciona correctament **des del punt de vista de l'usuari**.

El concepte de Test d'acceptació, entès com que prova que les funcionalitats funcionen com l'usuari espera, fa que siguin ortogonals a la piràmide de tests. Així, tot i que s'acostumen a incloure en la part més alta, també poden existir en nivells més baixos.

Estratègies de prova

Tests d'Exploració

Són proves de Caixa Negra

Són manuals.

Permeten provar situacions extranyes i provocar errors en l'aplicació.