Testning

1DV404, HT14

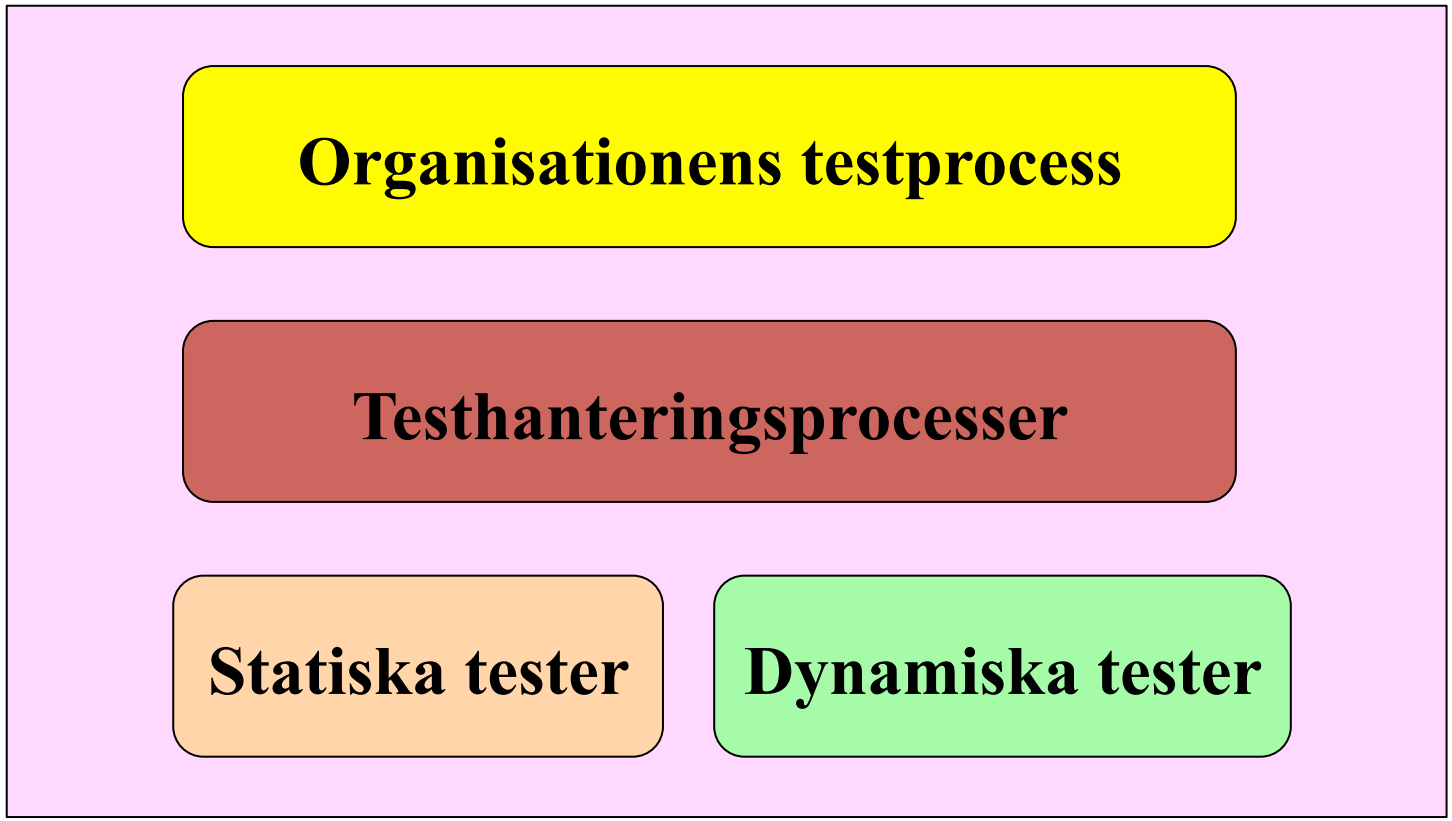
Jesper Andersson

Kap **21 + Testing Primer**

Testning

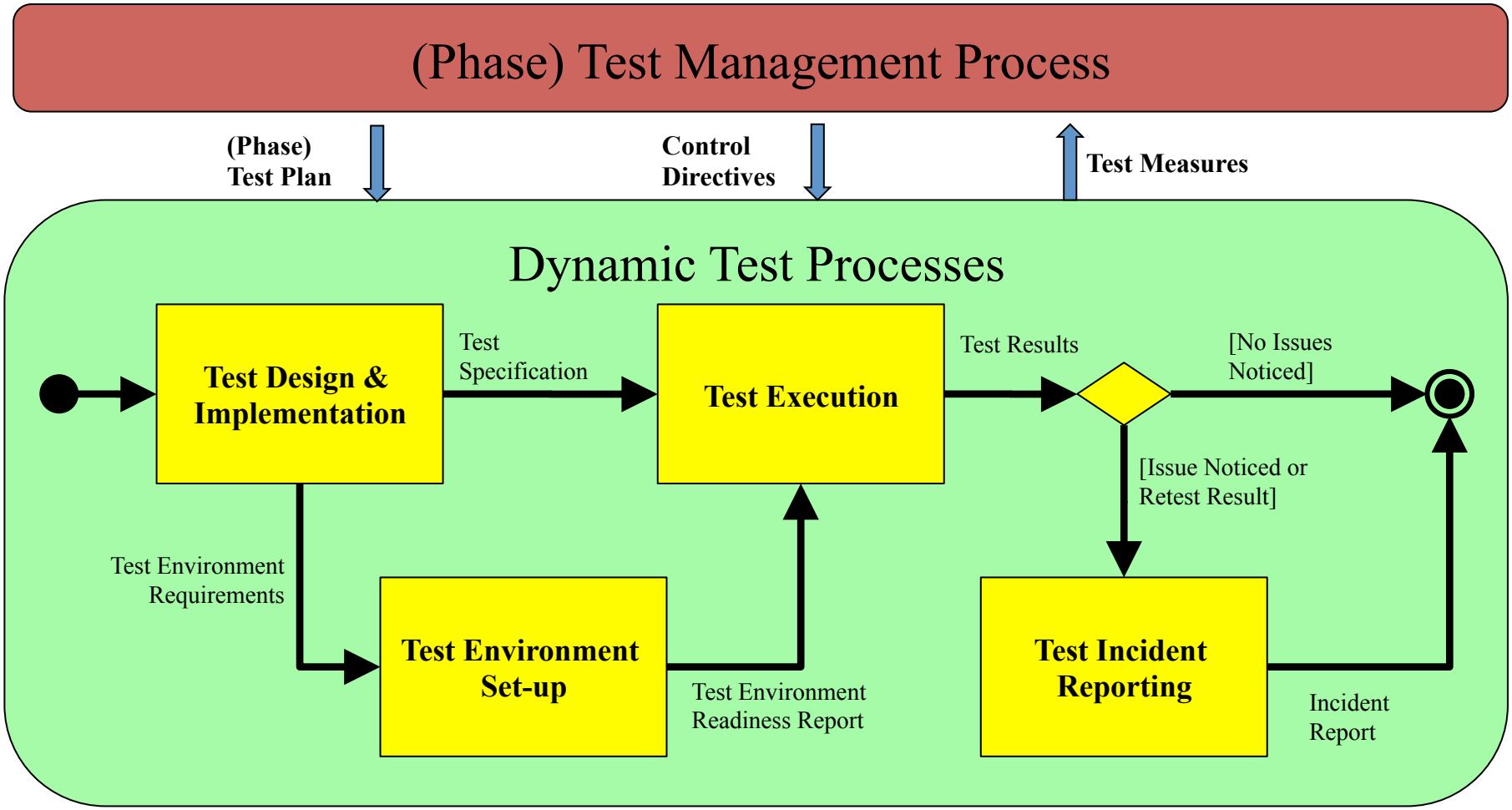
- ✓ Testningens huvudsakliga syfte är att **reducera risker.**
- ✓ Osäkerhetsfaktorer inom utvecklingen av ny programvara kan få ett projekt att spåra ur.
- ✓ Även mindre risker kan orsaka förseningar
- ✓ Genom att kontinuerligt testa och lösa påkomna problem kan man
  - Identifiera risknivåer.
  - Fatta väl underbyggda beslut
  - Därmed reducera osäkerhet och risker och få bort fel.

Testprocesser – ISO29119 (jmf. med UP)



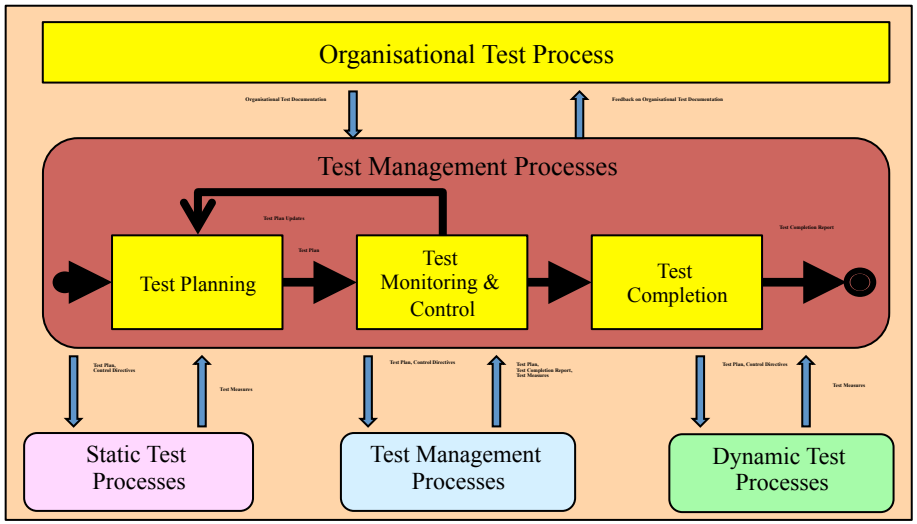
Integritet!!!

Dynamiska tester



Test Suite – “Testsvit”

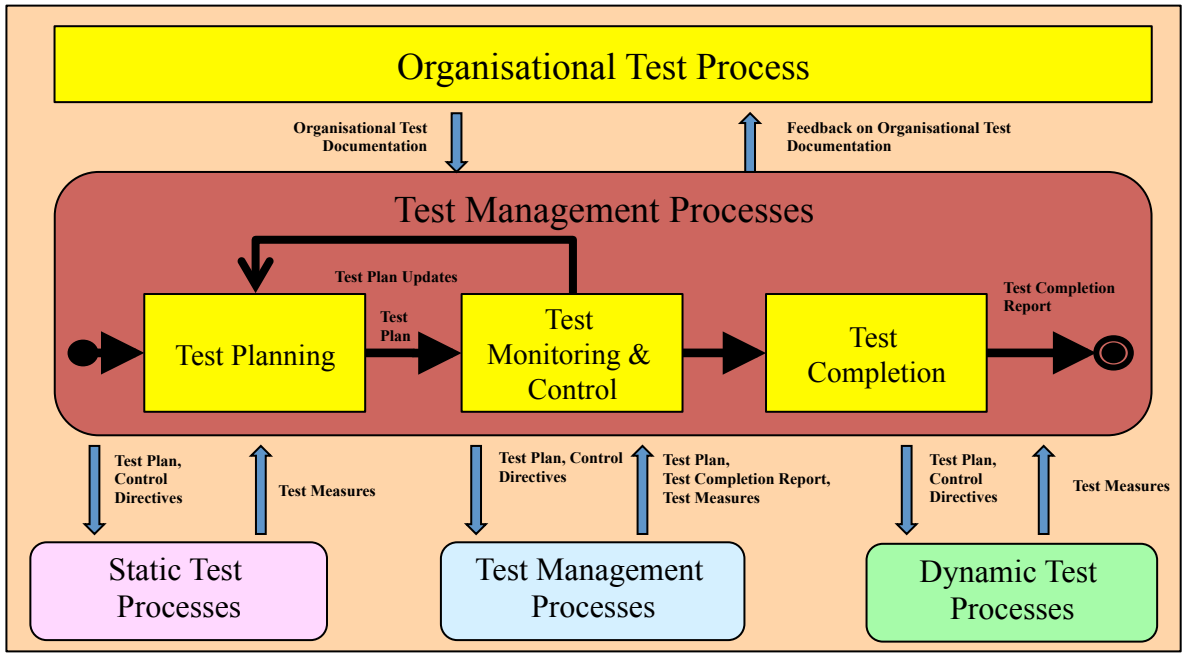
- ✓ En samling av *testfall* för att testa ett *mjukvarusystem*
- ✓ En testsvit innehåller
  - Testfall för olika objekt, mål och tekniker.
  - Information om hur man skall konfigurera SUT (System Under Test)
- ✓ Dessutom beskrivs hur systemomgivningen skall sättas upp



Testobjekt, målsättning & tekniker

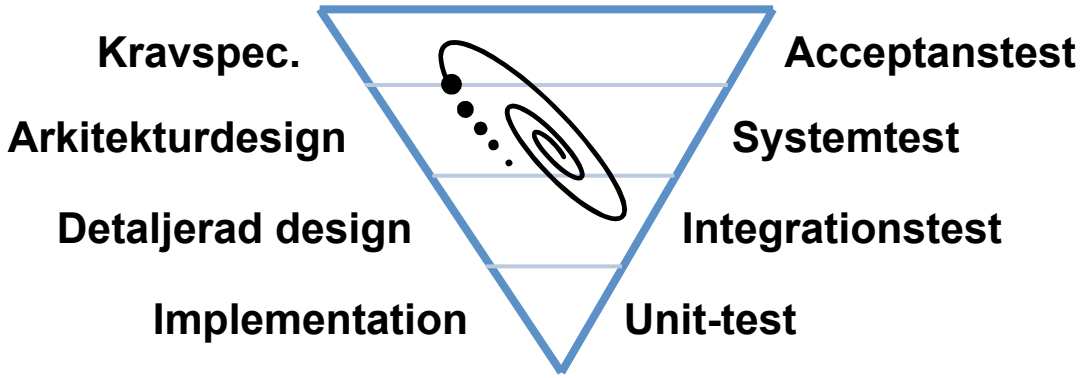
1. *Identifiera* - Test Object ➔ Vad!
2. *Definiera* - Test Objective ➔Varför!
3. *Välj* - Test Technique ➔Hur!

Testsvit



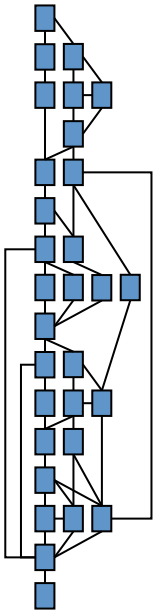
Test nivåer kopplat till faser

- ✓ I varje iteration kan vi testa på olika nivåer
- ✓ Unit-test kopplas oftast direkt till utvecklare
- ✓ Funktions/Integrationstester
- ✓ Systemtester
- ✓ Acceptanstest, sent i transition-fasen



Unittester (UT)

- ✓ UT, verifiera de minsta testbara enheterna i ett system.
- ✓ ***Test Object*** är ”unit” (typiskt en klass eller metod)
- ✓ ***Test Objective*** är att identifiera fel/defekter i koden.
- ✓ Dynamisktestning
  - UT använder ofta ”white box testning”.
  - Utförs därför oftast av utvecklare med direkt tillgång till koden
  - Man mäter ”kodtäckning” (*code coverage)*
- ✓ Statisk testning
  - Granskningar
  - Kodstandard



White-box testning – Täckningskriteria

- ✓ Alla-vägar(omöjligt)
- ✓ Instruktion
- ✓ Gren
- ✓ Villkor (Multi-villkor)

x >= 9	y > -3
T	T
T	F
F	T
F	F

```
int ex(int x, int y) {
    int z = 0;
    if ((x >= 9) && (y > -3)) {
        z = x++;
    }
    return z-22;
}
```

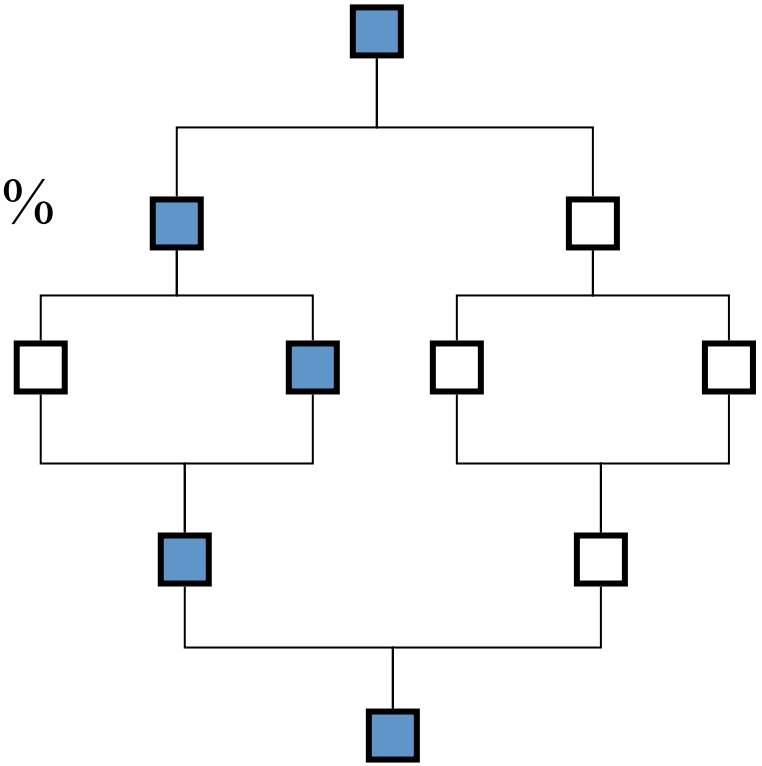

Täckning - Example

```
if …
   if …
   else …
else …
   if …
   else …
endif
```

Instruktion 5/10, 50%

Gren 2/6, 33%

Väg 1/4, 25%



Mäta täckningsgrad – Instrumentering

- ✓ Instrumentering är en teknik som lägger till funktionalitet som endast används under utvecklingsfasen.
  - Vanligast är att man instrumenterar källkod
  - Även möjligt att instrumentera genererad kod, exempelvis binärkod
- ✓ Instrumenteringar kan läggas till tas bort allt eftersom behov finns.

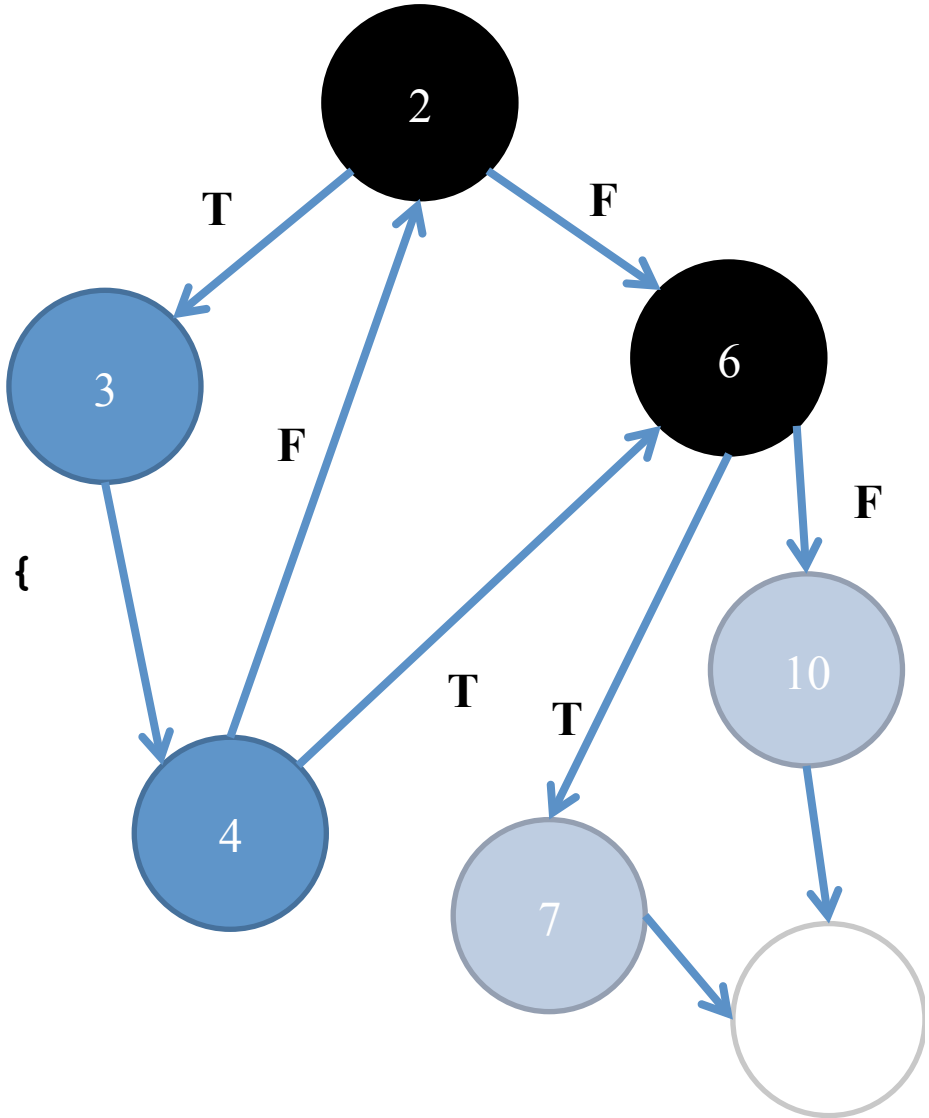
- ✓ **Varför?**
  - Påverkar exempelvis prestanda negativt!
  - Komplicerar felsökning.

Anropsgraf (Call-graph)

- ✓ En grafnotation för alla möjliga exekveringsvägar.
- ✓ En graf består av noder och bågar (kanter)
- ✓ Varje nog representerar ett så kallat “basic block” (basblock)
  - Basblock: en följd av kod utan hopp (ut) eller hopp mål (in)
    - Mål påbörjar ett block
    - Hopp avslutar ett block.
  - Riktade bågar representerar hopp i anropsflödet.
  - Två speciella block:
    - Ingångsblock, genom vilket programflödet påbörjas i anropsgrafen
    - Exitblock, genom vilket allt flöde lämnar anropsgrafen.

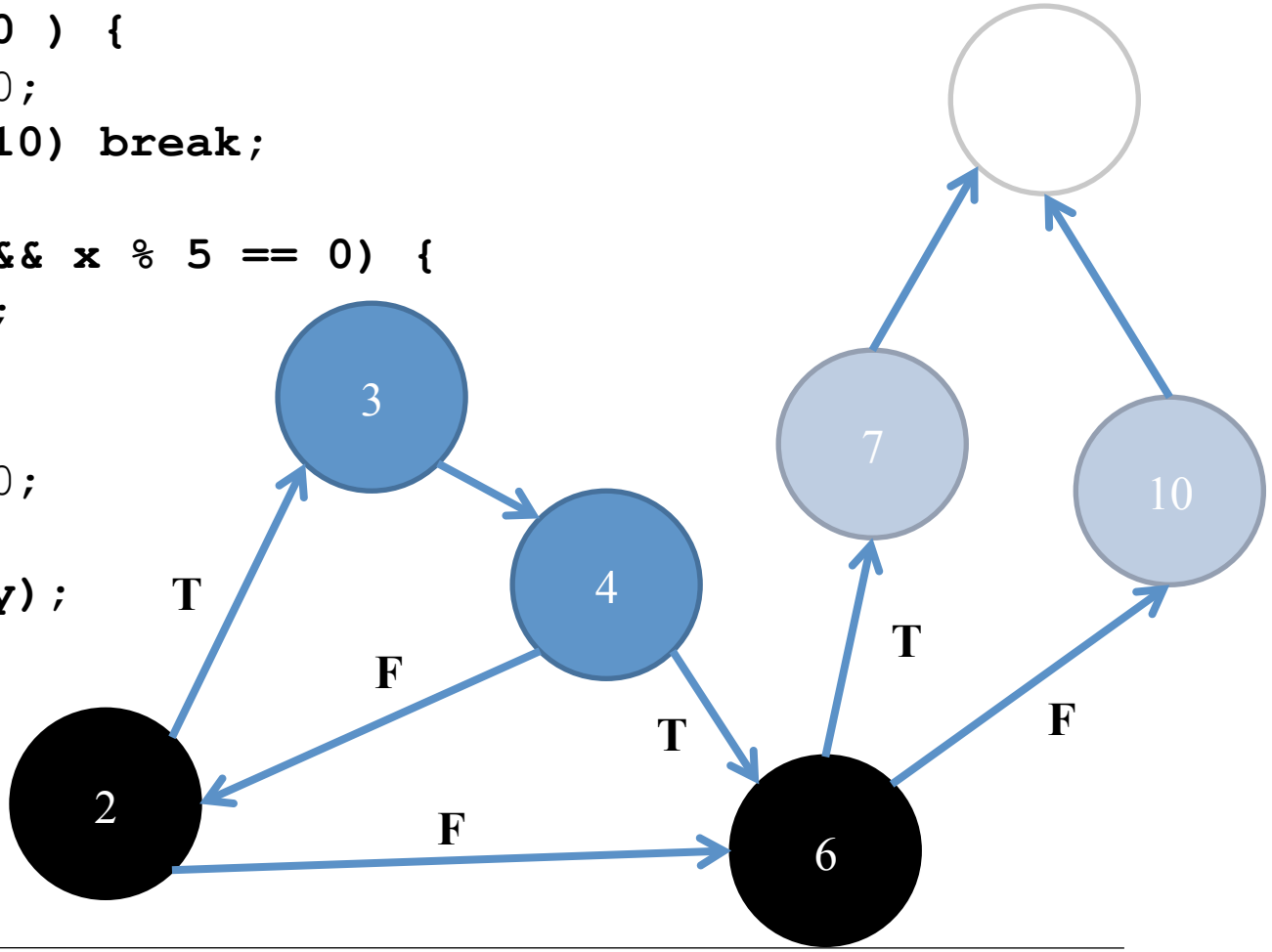
Call-graph (Anropsgraf)

```
1 public int p(int x, int y) {
2   while (x > 10 ) {
3       x = x - 10;
4       if (x == 10) break;
5   }
6   if ( y < 20 && x % 5 == 0) {
7       y = y+ 20;
8   }
9   else {
10      y = y - 20;
11  }
12  return 4*(x+y);
13}
```



Exempel – CFG, Instruktionstäckning

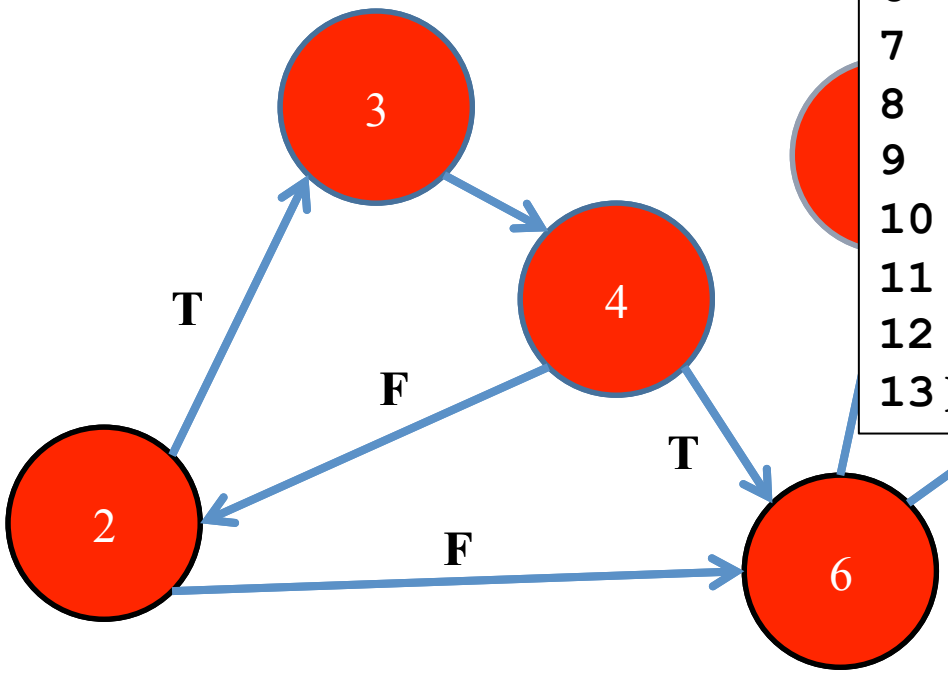
```
1 public int p(int x, int y) {
2   while (x > 10 ) {
3       x = x - 10;
4       if (x == 10) break;
5   }
6   if ( y < 20 && x % 5 == 0) {
7       y = y+ 20;
8   }
9   else {
10      y = y - 20;
11  }
12  return 4*(x+y);
13}
```



Testfall 1: Instruktionstäckning

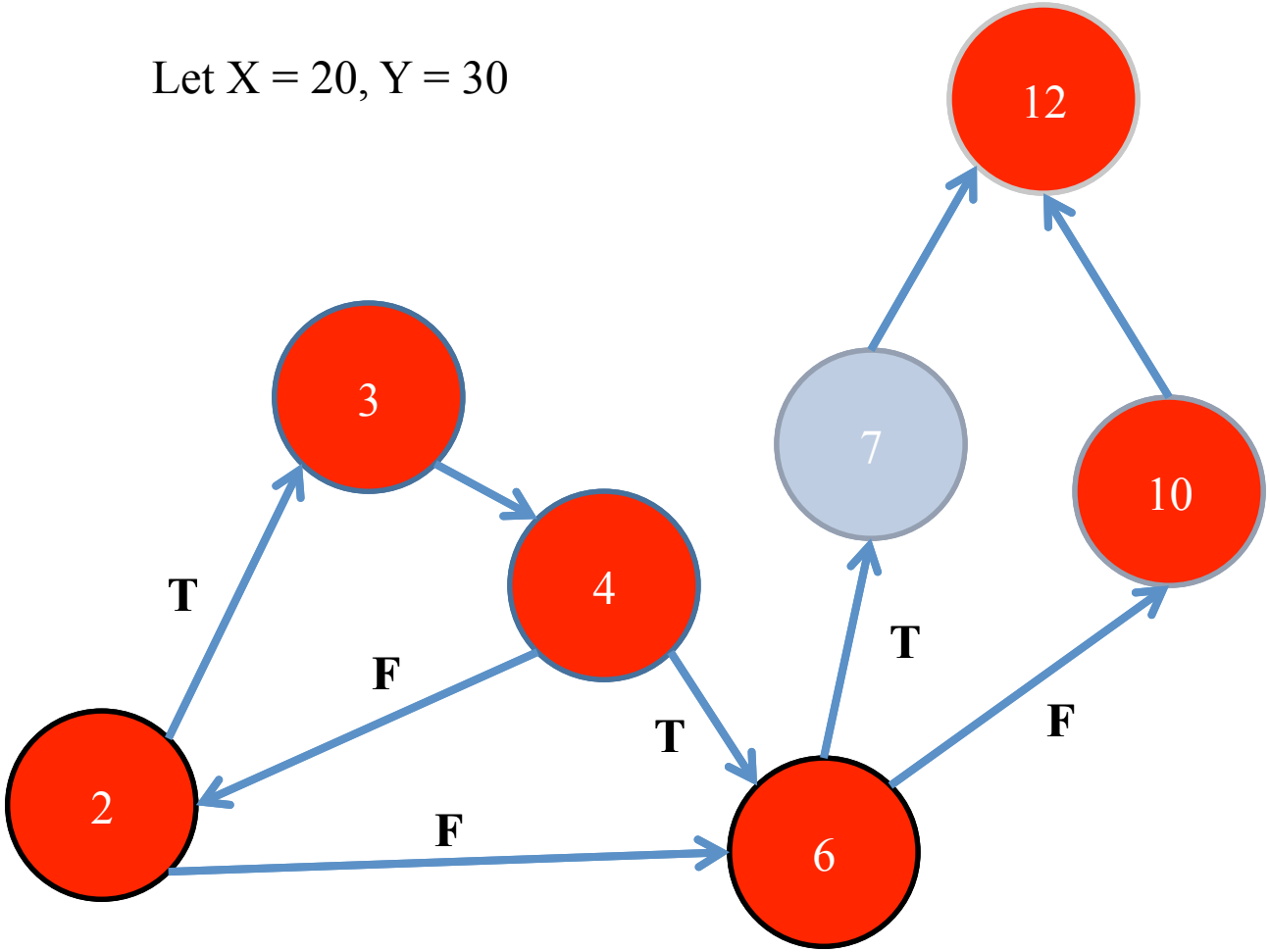
Let X = 20, Y = 10

```
1 public int p(int x, int y) {
2   while (x > 10 ) {
3       x = x - 10;
4       if (x == 10) break;
5   }
6   if ( y < 20 && x % 5 == 0) {
7       y = y+ 20;
8   }
9   else {
10      y = y - 20;
11  }
12  return 4*(x+y);
13}
```



Testfall 2: Instruktionstäckning

Let X = 20, Y = 30

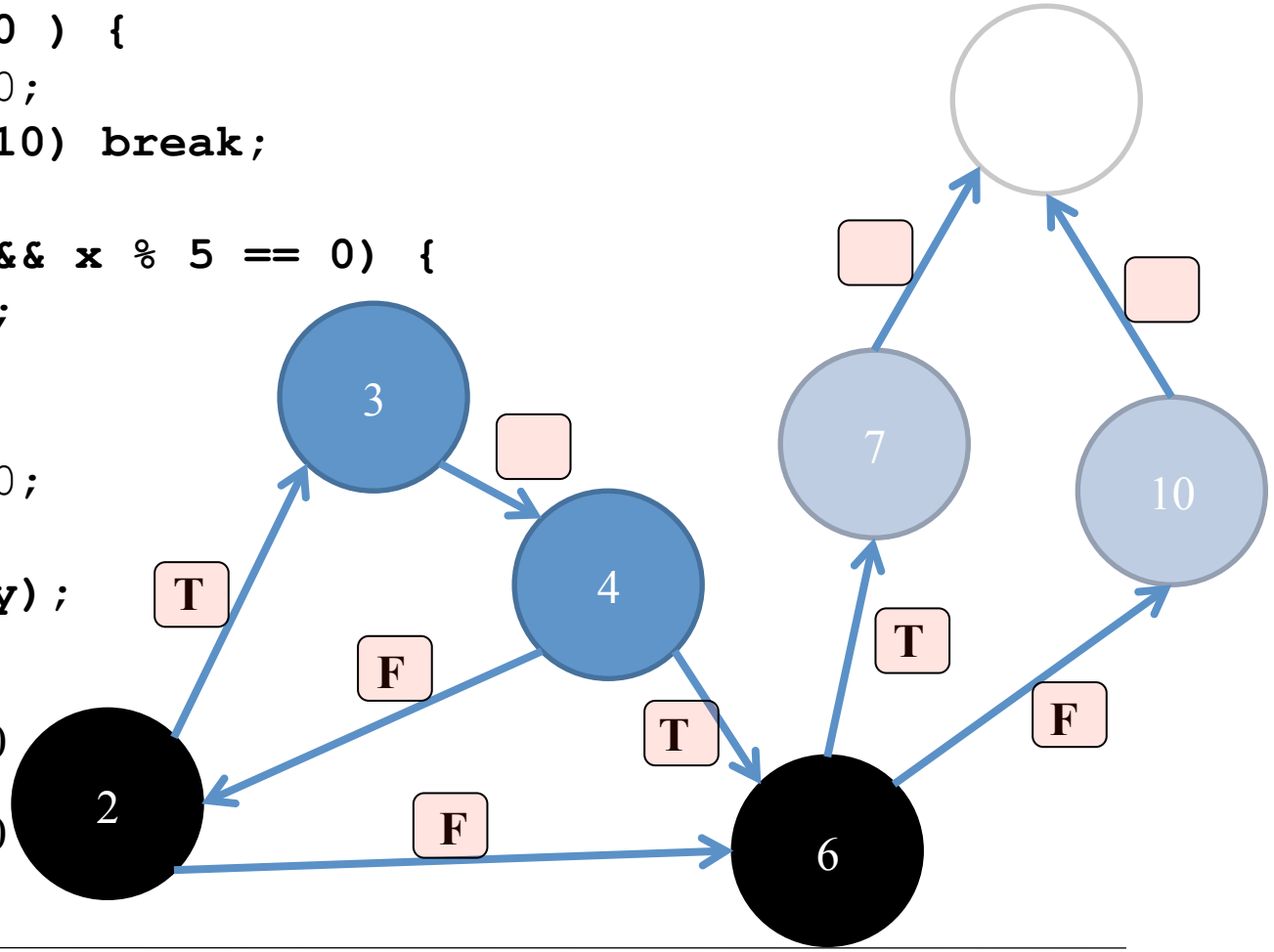


Exempel – CFG, Grentäckning (Branch)

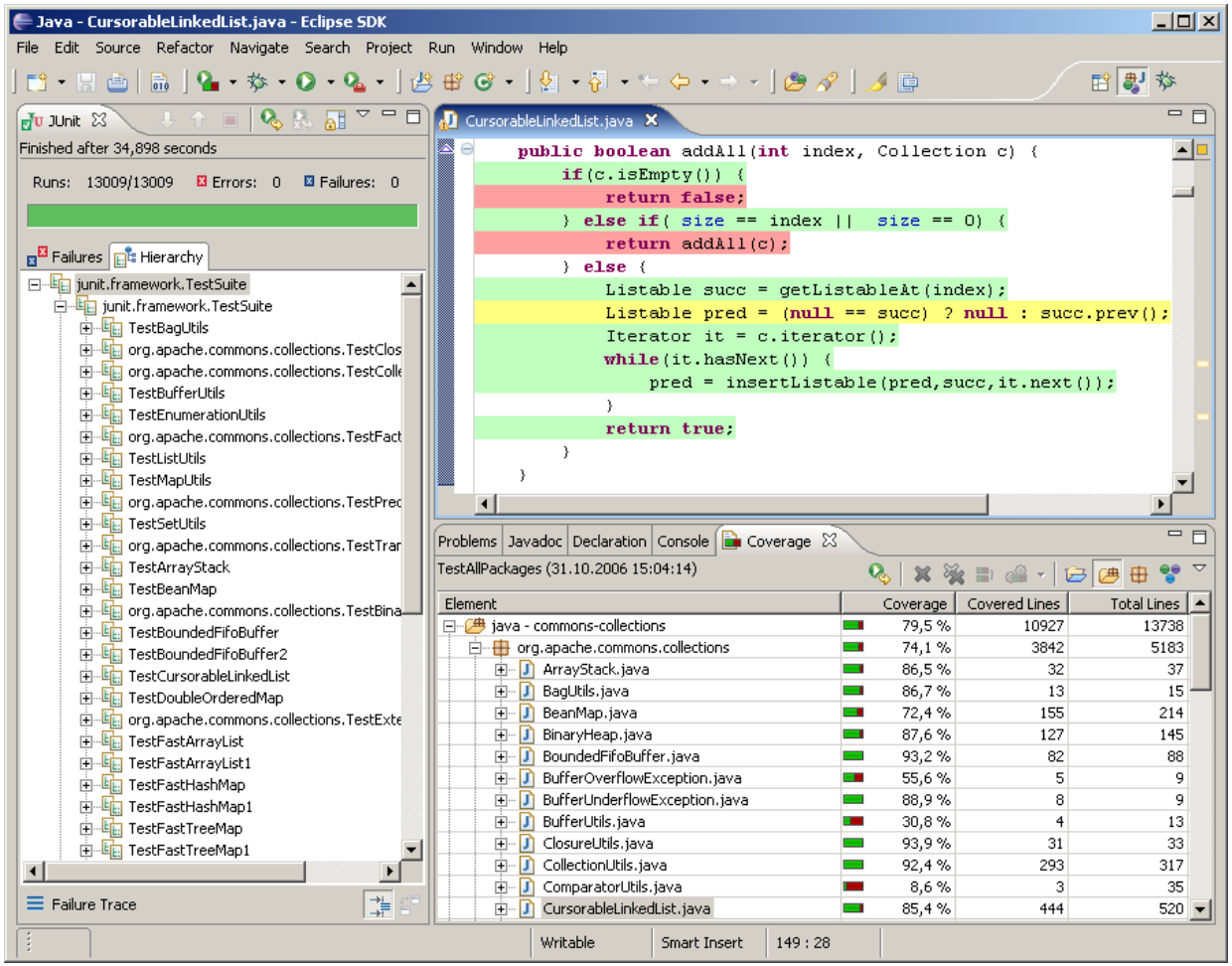
```
1 public int p(int x, int y) {
2   while (x > 10 ) {
3       x = x - 10;
4       if (x == 10) break;
5   }
6   if ( y < 20 && x % 5 == 0) {
7       y = y+ 20;
8   }
9   else {
10      y = y - 20;
11  }
12  return 4*(x+y);
13}
```

Testfall 1, Let X = 20, Y = 10

Testfall 2, Let X = 20, Y = 30



Täckningsverktyg – EclEmma



Exempel– EclEmma

- ✓ Visar om källkoden exekverats eller ej
- ✓ Grön – Fullständigt
- ✓ Gult – delvis
- ✓ Rött – Inte alls

```
public boolean addAll(int index, Collection c) {
    if(c.isEmpty()) {
        return false;
    } else if( size == index ||  size == 0) {
        return addAll(c);
    } else {
        Listable succ = getListableAt(index);
        Listable pred = (null == succ) ? null : succ.prev();
        Iterator it = c.iterator();
        while(it.hasNext()) {
            pred = insertListable(pred,succ,it.next());
        }
        return true;
    }
}
```


xUnit – Arkitektur för Testramverk

- ✓ Testningsramverk som används för att skriva och utföra upprepade *automatiska tester*
- ✓ Ger en struktur för att skriva "*testdrivare*"
- ✓ xUnit inkluderar:
  - *Assertions* för att testa förväntade resultat
  - Möjligheter att dela "gemensam testdata", testfixturer
  - Möjlighet att skapa testsviter
  - Grafiska och textbaserade "testrunners"

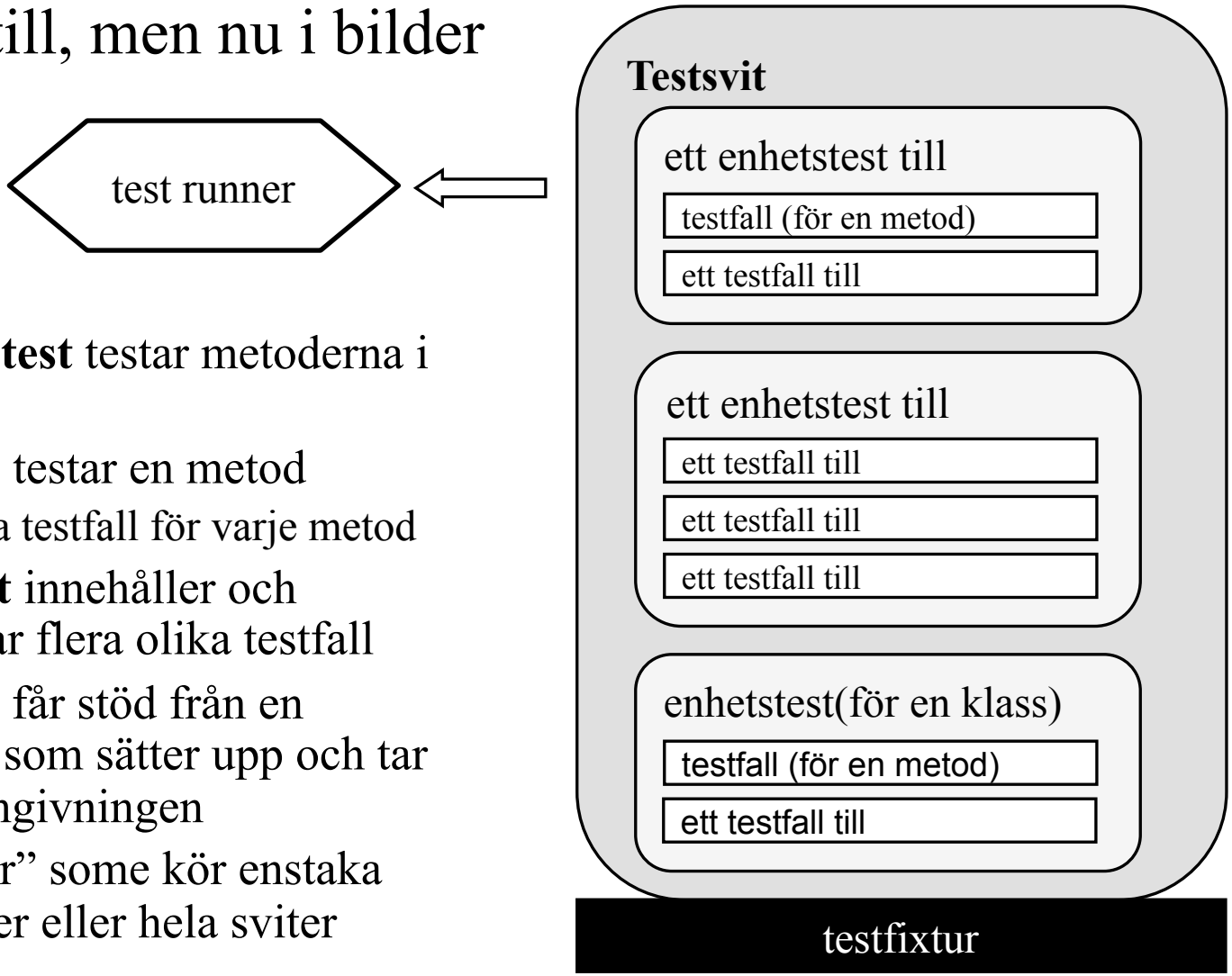
- ✓ Exempel,
  - JUnit – junit.org
  - xUnit – xunit.github.io

xUnit Arkitekturen – Tester

- ✓ xUnit används för att testa …
  - … ett helt testobjekt (t.ex. en klass)
  - … en del av ett objekt (t.ex. en metod eller ett antal samverkande metoder)
  - … interaktion mellan flera objekt (Integrationstestning)
- ✓ En testklass innehåller fler än ett test
  - Varje test skrivs i en testmetod
- ✓ Testklassen omfattar även :
  - "Testrunners" för att exekvera testerna ("main()")
  - En uppsättning testmetoder (test cases) med assertions
  - Metoder för att (fixtures)
    - korrigera tillståndet före ett test
    - uppdatera tillståndet efter varje test
    - och/eller efter alla tester

    } Sequencing
- ✓ Mer information finns på junit.org

En gång till, men nu i bilder



- ✓ Ett **enhetstest** testar metoderna i en klass
- ✓ Ett testfall testar en metod
- ✓ Du har flera testfall för varje metod
- ✓ En **testsvit** innehåller och kombinerar flera olika testfall
- ✓ Testsviten får stöd från en **testfixtur** som sätter upp och tar ned testomgivningen
- ✓ “testrunner” some kör enstaka enhetstester eller hela sviter

Att skriva tester för xUnit – exempel JUnit

- ✓ @test
- ✓ Använd metoder i klassen `junit.framework.assert`
- ✓ Varje metod kontrollerar ett villkor (assertion) och återrapporterar till "testrunnern" om ett test gick fel eller inte

- ✓ "Testrunnern" använder resultatet för att rapportera tillbaka till användaren.
- ✓ Alla testmetoder returnerar `void`
- ✓ Exempel på några metoder i junit.framework.assert
  - `assertTrue (boolean)`
  - `assertTrue (String, boolean)`
  - `assertEquals (Object, Object)`
  - `assertNull (Object)`
  - `Fail (String)`

Att ta fram testfall från användningsfall

Orginal: Chris Collins, *www.christophertcollins.com/presentations/Understanding**UseCases**.ppt*

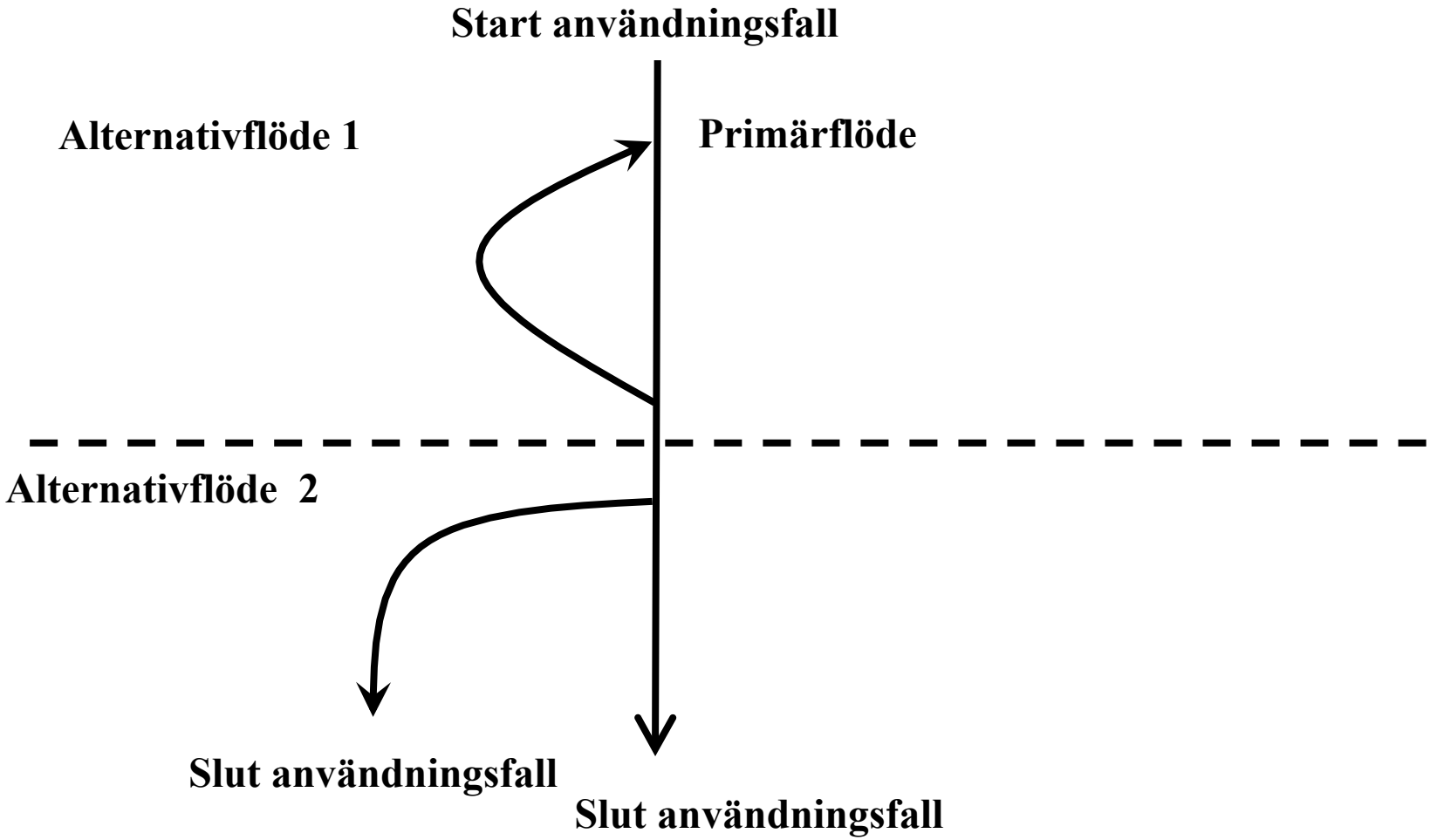
- ✓ Liknar whiteboxtestning och siktar på hög täckningsgrad
- ✓ En process i fyra steg
  - **Steg 1**: Identifiera **vägarna** genom ett användningsfall – {Scenario}
  - **Steg 2**: Identifiera testfallen – ett eller flera för varje scenario
  - **Steg 3**: Identfiera testfallen
  - **Steg 4**: Lägg till testdata så att testfallen blir kompletta

Exempel – Användningsfall “Lås upp skärmen”

Unlock Screen

- ✓ Primärflöde
  1. The user selects unlock command
  2. System brings up logon screen
  3. The user enters valid user Id and password
  4. The user selects to logon
  5. The system unlocks the screen
- ✓ Alternativflöde 1 – Invalid Password
  - 3a. The user enters valid user Id and invalid password
  - 4a. The user selects to logon to the system
  - 5a. The system indicates error logging on and returns to logon screen
- ✓ Alternativflöde 2 – Cancel
  - 3b. The user select to cancel
  - 4b. The system does not log user on and returns locked screen

Steg 1: Scenarios för användningsfallet



Steg 1: Scenariomatris – Identifiera scenarios

Scenario #	Ursprungsflöde	Alternativflöde	Nästa Alternativ	Nästa Alternativ
1	Primärflöde			
2	Primärflöde	Alternativflöde 1		
3	Primärflöde	Alternativflöde 1	Alternativflöde 2	
4	Primärflöde	Alternativflöde 2		

Exempel – Användningsfall “Lås upp skärmen”

Unlock Screen

- ✓ Primärflöde
  1. The user selects unlock command
  2. System brings up logon screen
  3. The user enters valid user Id and password
  4. The user selects to logon
  5. The system unlocks the screen
- ✓ Alternativflöde 1 – Invalid Password
  - 3a. The user enters valid user Id and invalid password
  - 4a. The user selects to logon to the system
  - 5a. The system indicates error logging on and returns to logon screen
- ✓ Alternativflöde 2 – Cancel
  - 3b. The user select to cancel
  - 4b. The system does not log user on and returns locked screen

Steg 2: Testfall

Test Case Id	Scenario	Unlock Screen Cmd	UserId	Pwd	Logon Cmd	Expected Result	Actual Result
1	Scenario 1						
2	Scenario 2						
3	Scenario 4						

Steg 3: Testvillkor

Test Case Id	Scenario	Unlock Screen Cmd	UserId	Pwd	Logon Cmd	Expected Result	Actual Result
1	*Scenario 1*	Valid	Valid	Valid	Yes	Logon	
2	*Scenario 2*	Valid	Valid	Invalid	Yes	Failure Return to logon	
3	*Scenario 4*	Valid	N/A	N/A	No	Return to Screen Lock	

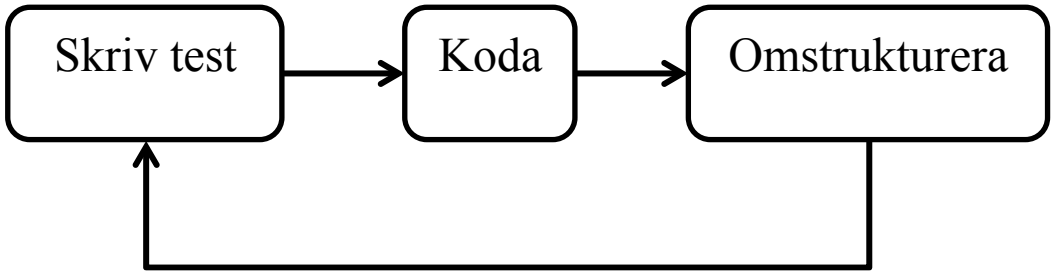
Steg 4. Lägg till testdata till testfallen

Test Case Id	Scenario	Unlock Screen Cmd	UserId	Pwd	Logon Cmd	Expected Result	Actual Result
1	*Scenario 1*	Ctr-atl-del	ctc	abc	Ok Btn	Logon	Logged on
2	*Scenario 2*	Ctr-atl-del	ctc	abd	Ok Btn	Failure Return to logon	Returned to Logon
3	*Scenario 4*	Ctr-atl-del	N/A	N/A	Cancel Btn	Return to Screen Lock	Returned to Screen lock

Test Driven Development

> If it's worth building, it's worth testing.
> If it's not worth testing, why are you wasting your time working on it?

- ✓ Testet ugör en design*specifikation*
  - *Pre och Post Villkor*
  - *Metodsignatur*
- ✓ Tvingar designern att tänka **"hur skall den användas"** innan **"hur skall den impelmenteras"**



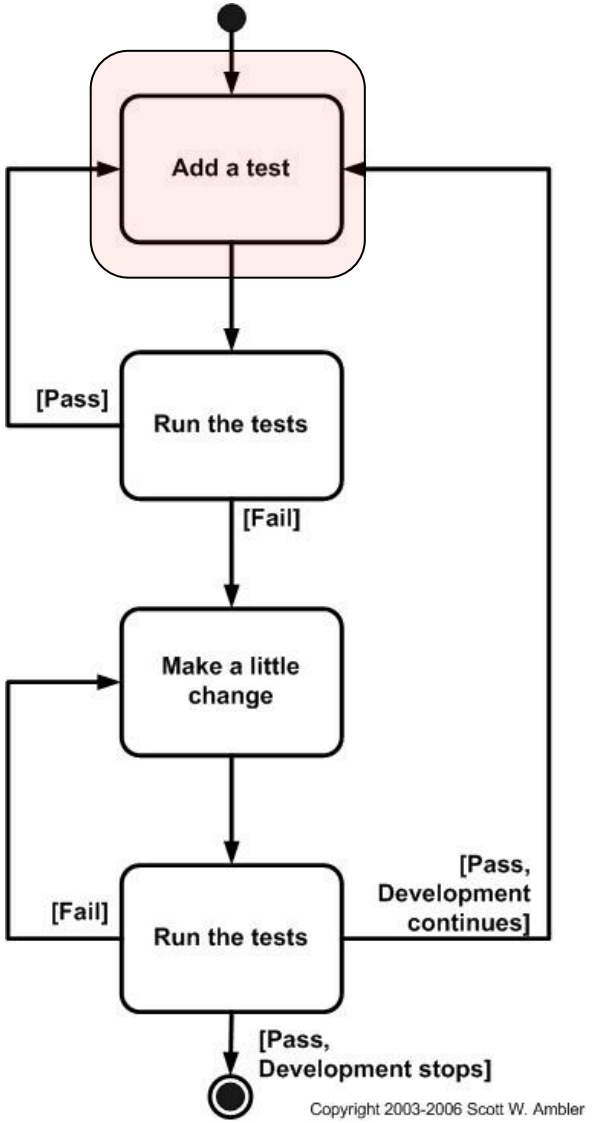
TDD handlar *inte om testning*, det handlar om *design*!

Test First Development

- ✓ En av det grundläggande principerna
- ✓ Detta är **ingen process**

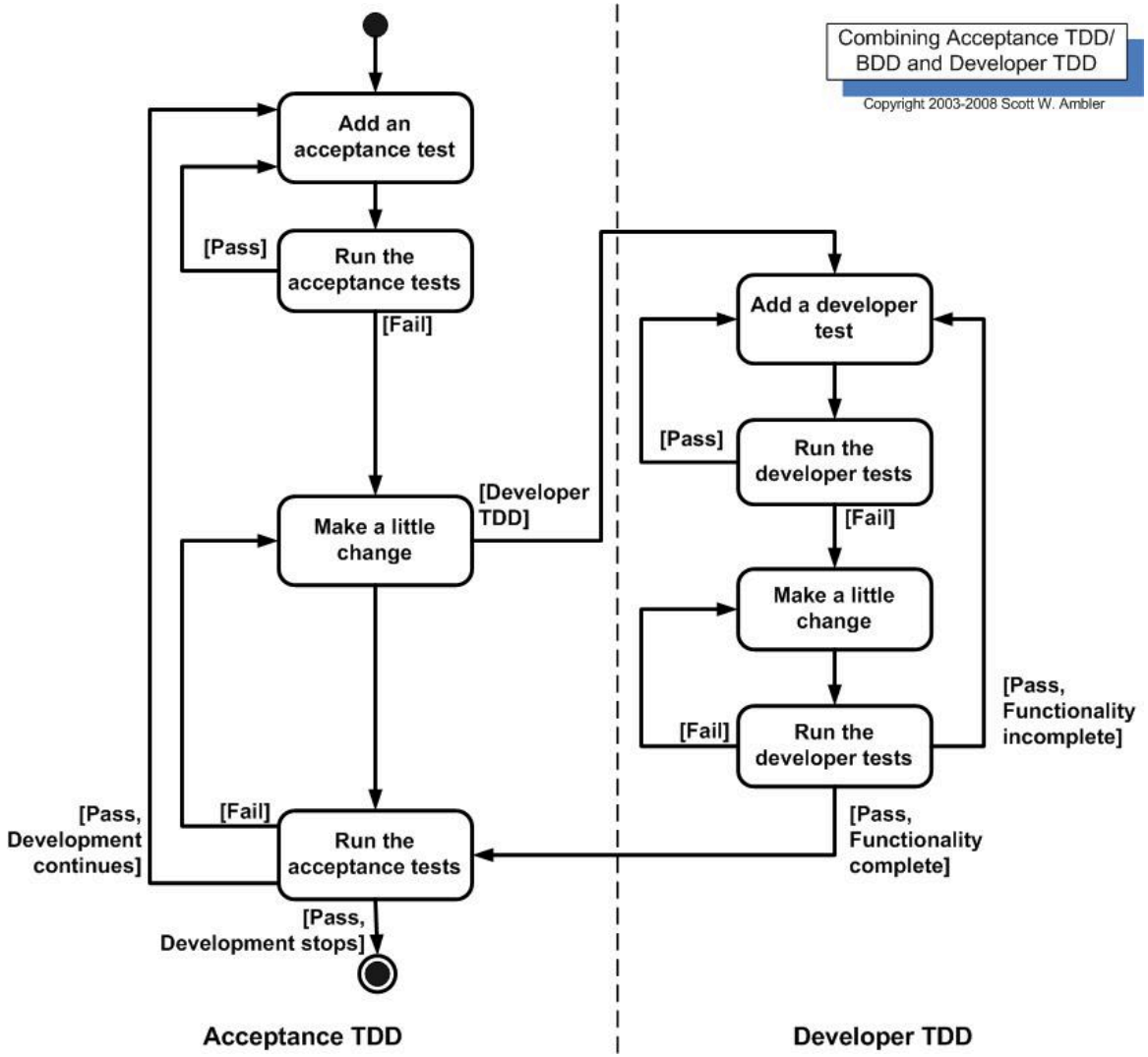
- ✓ Är en **princip** (practice) inom ”agile development”

- ✓ Kan användas för att skapa hierarkiska design strukturer



TDD – Två nivåer

- ✓ Acceptansnivå
  - Högnivå
  - Integration
  - Specifikation för
- ✓ Utvecklarnivå
  - Detaljeradnivå
  - "Unit nivå"

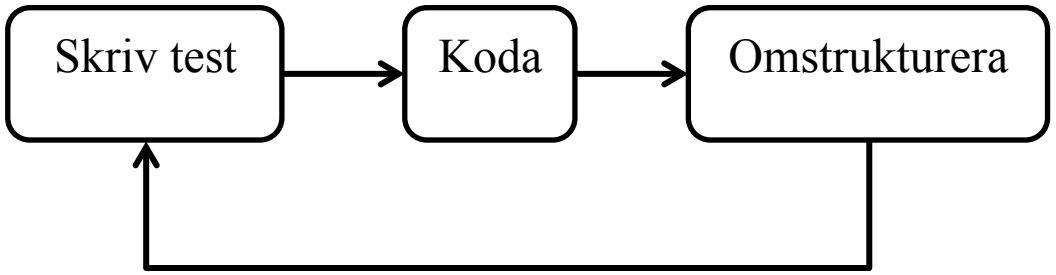


Acceptans-TDD (ATDD).

- ✓ Skriv ett acceptans-test (eller behavioral specification)
- ✓ Sedan skriver du precis tillräckligt med kod för att testet skall gå igenom
- ✓ Målet med acceptans-TDD är att skapa exekverbara specifikationer
- ✓ Kallas även *Behavior Driven Development* (BDD).

Omstrukturering (Refactoring)

- ✓ Ändra strukturen på koden utan att förändra beteendet
- ✓ Exempel:
  - Ändra namn
  - Bryt ut metod eller interface
  - Ta bort metodanrop (inline)
  - Pull up /Push down
    - Flytta en metod från en subklass till en superklass
    - Flytta en metod från en superklass till en subklass
- ✓ De flesta IDEer (t.ex. Eclipse) erbjuder automatiska omstruktureringar



Denna vecka

- ✓ Torsdag – SCRUM