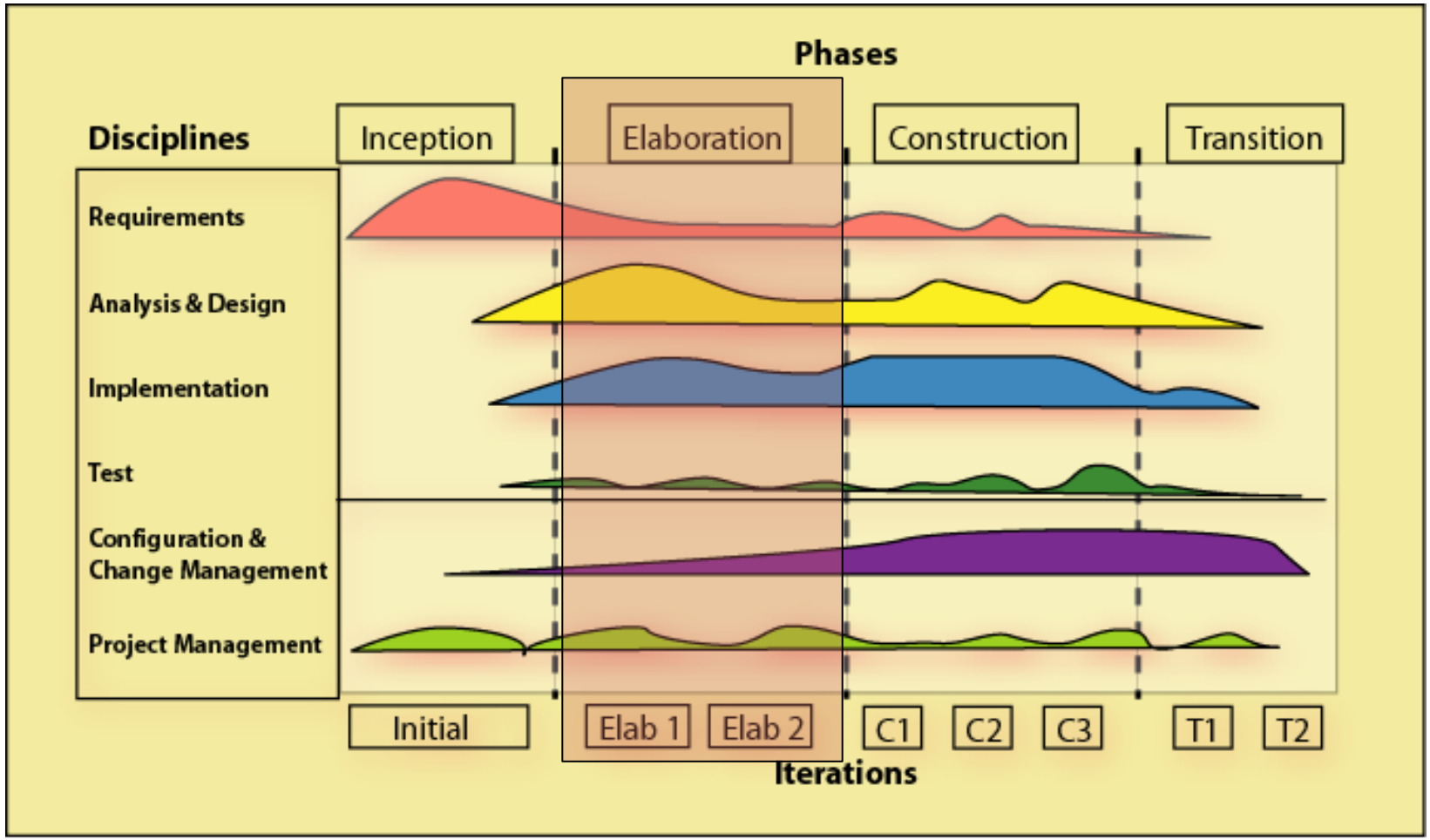
Testning

1DV404, HT14

Jesper Andersson

Kap **21 + Testing Primer**

UP – Faser


Phases
Disciplines
Inception
Elaboration
Construction
Transition
Requirements
Analysis & Design
Implementation
Test
Configuration &
Change Management
Project Management
Initial
Elab 1
Elab 2
C1
C2
C3
T1
T2
Iterations

Elaboration

- ✓ **Syfte:** Fastställa och validera en basarkitektur för systemet vilket ger en stabil grund för den största delen av utvecklingsarbetet i nästa fas.
- ✓ Krav
  - Få en mer detaljerad förståelse av kraven.
  - Få en fördjupad förståelse för de kritiska krav som valideras av arkitekturen ➔ T. ex. förstå hur de samverkar
- ✓ Design
  - Designa, implementera, och ***validera*** ➔ fastställa basarkitekturen.
  - Basarkitektur: Ett skelettsystem, med “kritisk funktionalitet”
- ✓ Riskhantering
  - Mildra de mest kritiska riskerna
  - Följd av **tester** av arkitekturen.

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.

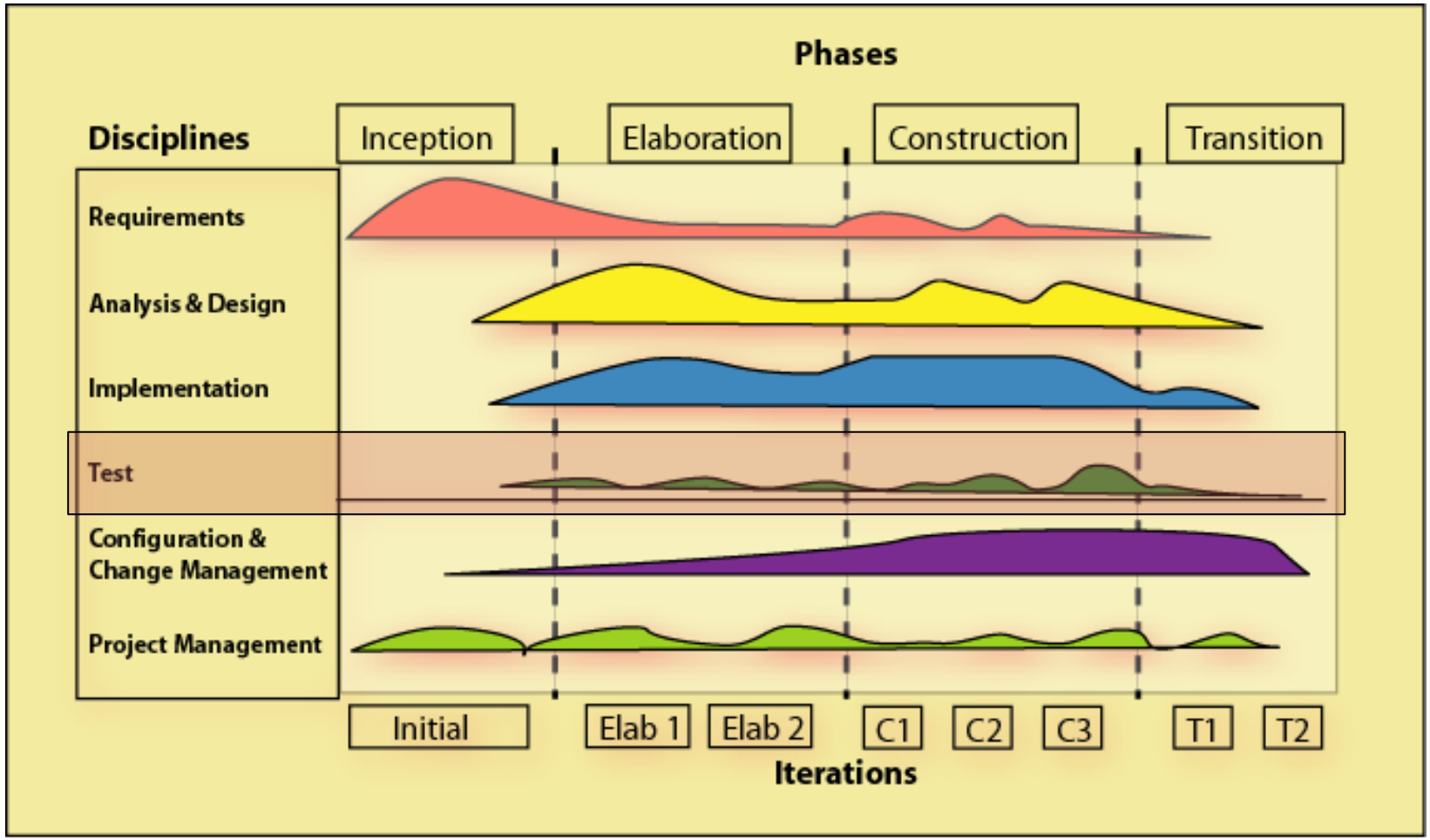
Verifiering & Validering

- ✓ Testning kan delas in i *Validering* och *Verifiering*
- ✓ Validering
  - Validering är en extern **Kvalitetssäkringsprocess**
  - Fråga till kunden: "*Bygger vi rätt system?"*

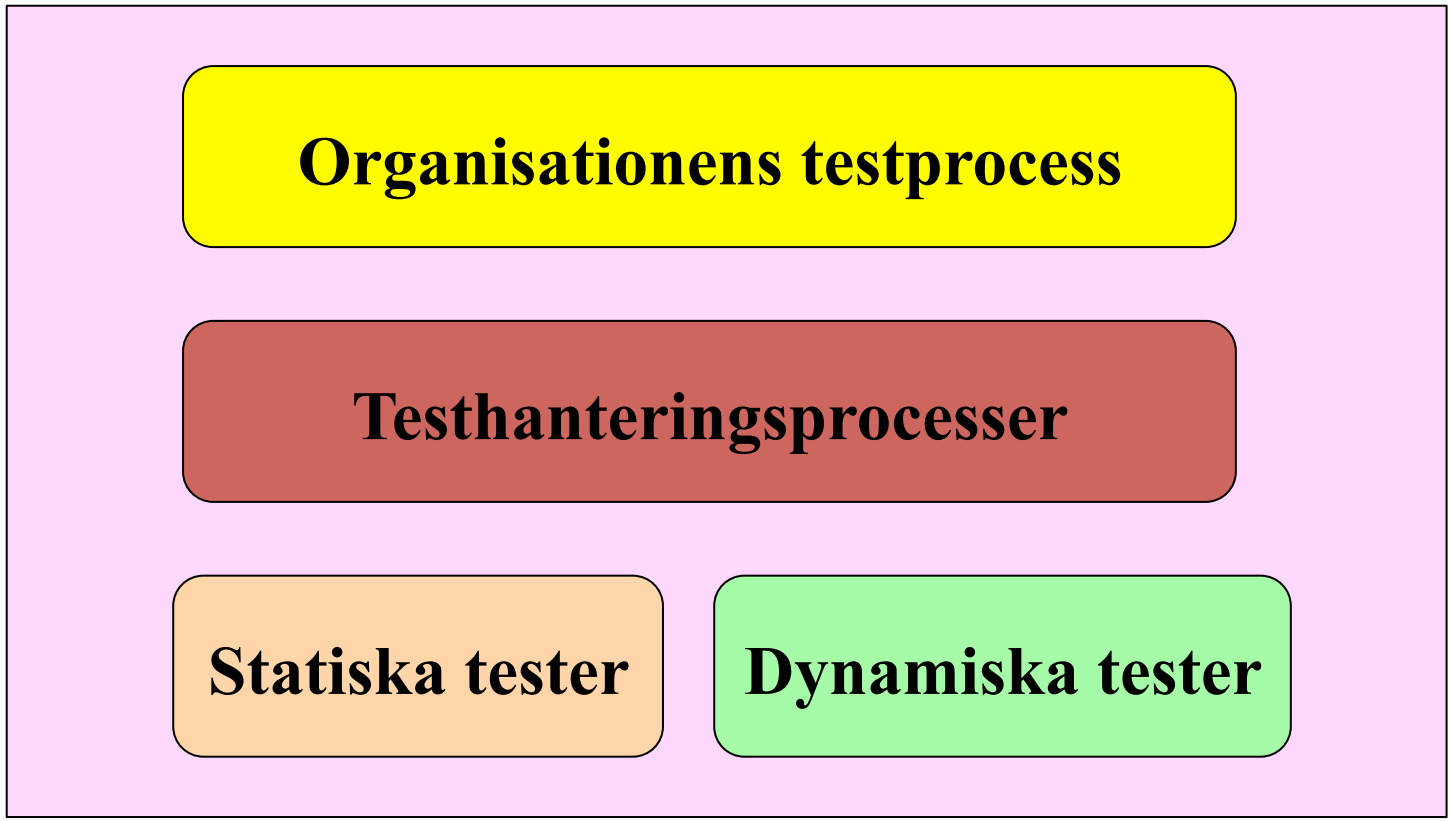
- ✓ Verifiering
  - Verifiering är en intern **Kvalitetskontroll**
  - Fråga: *"Bygger vi systemet rätt?"*

Exempel: Verifiering kontrollerar att något är "internt korrekt", exempelvis verifierar testning att mjukvaran är korrekt enligt kravspecifikationen", men inte att kraven i sig är de rätta!

UP – Testning

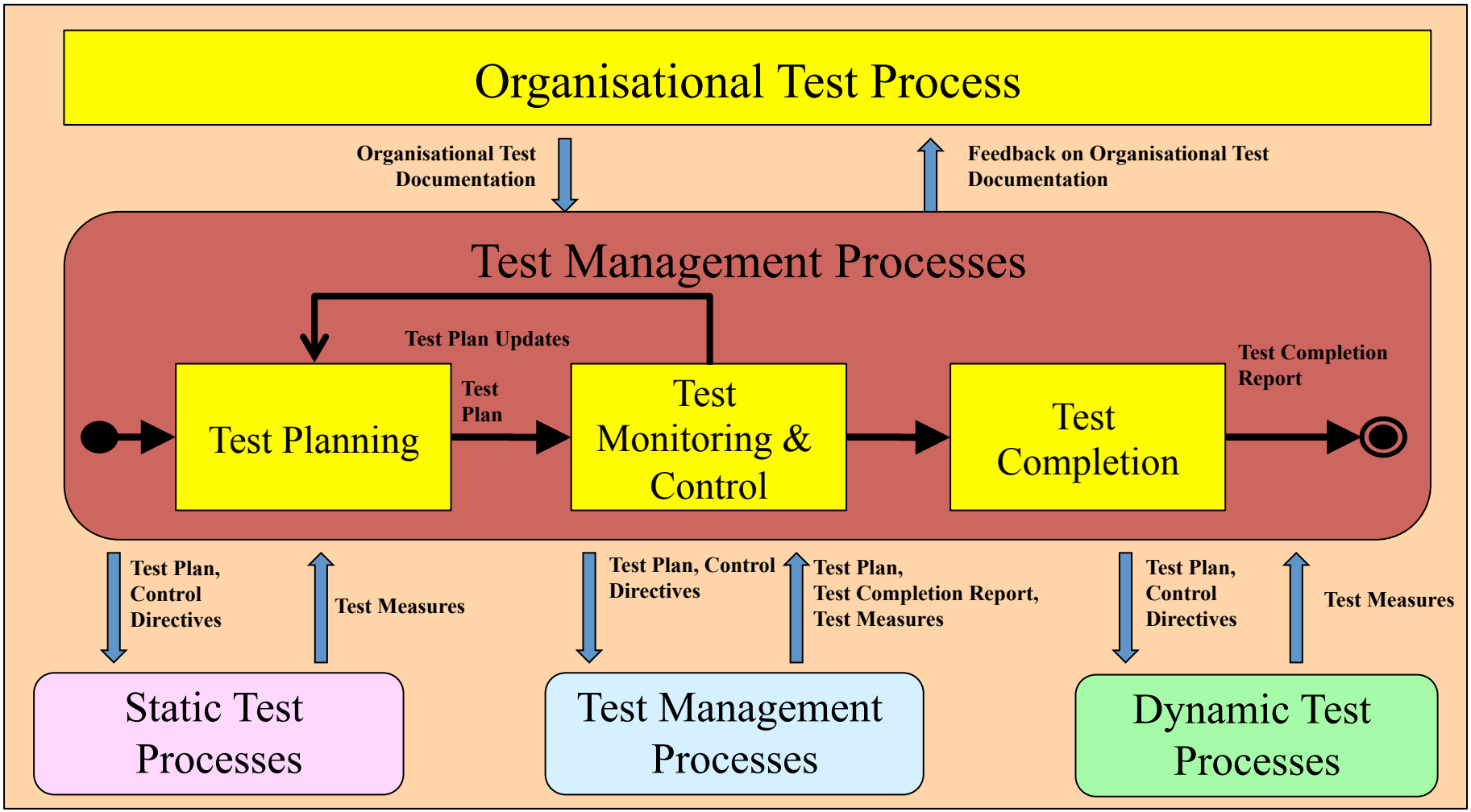

Phases
Disciplines
Inception
Elaboration
Construction
Transition
Requirements
Analysis & Design
Implementation
Test
Configuration & Change Management
Project Management
Initial
Elab 1
Elab 2
C1
C2
C3
T1
T2
Iterations

Testprocesser – ISO29119 (jmf. med UP)

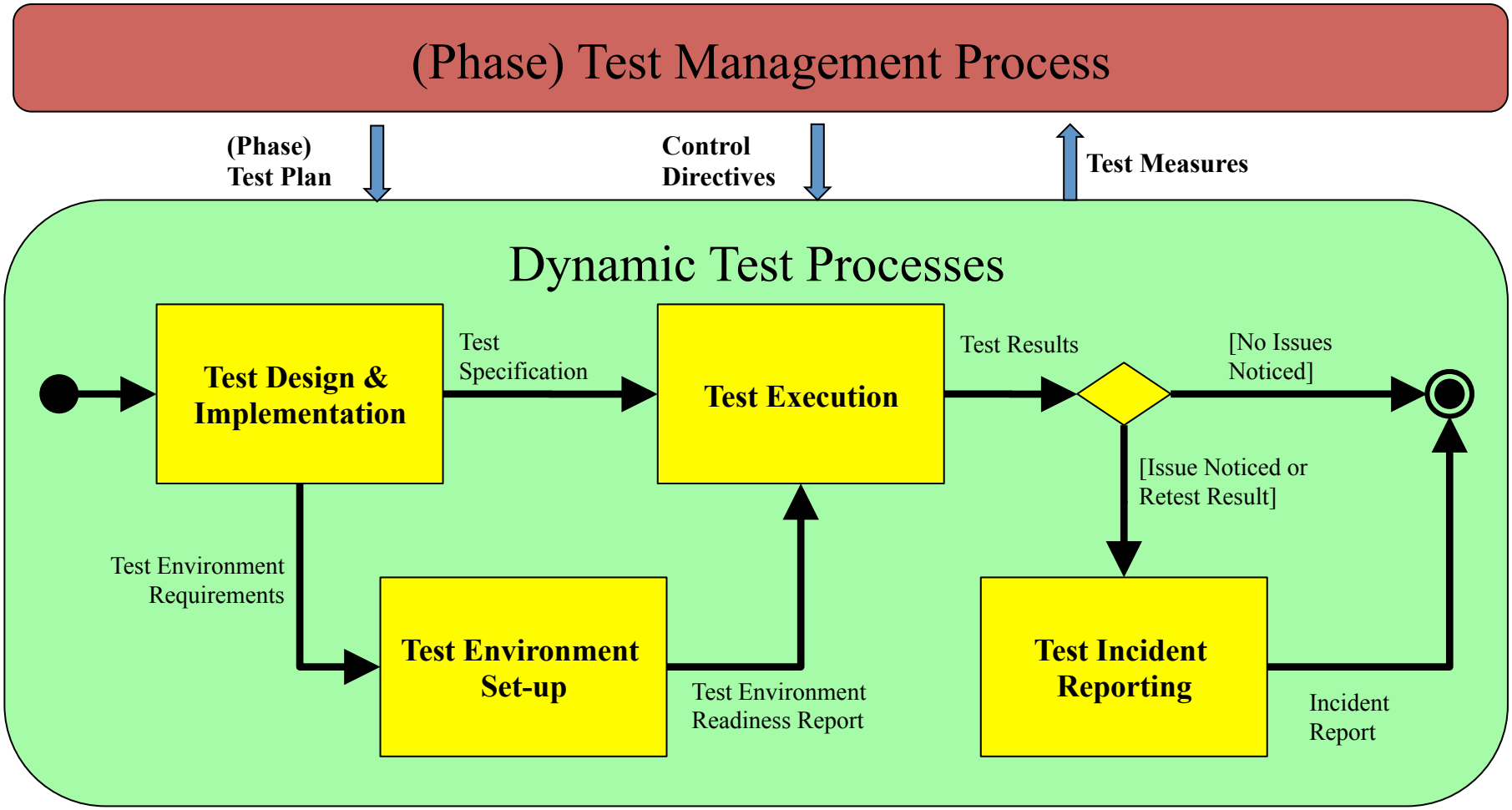


Integritet!!!

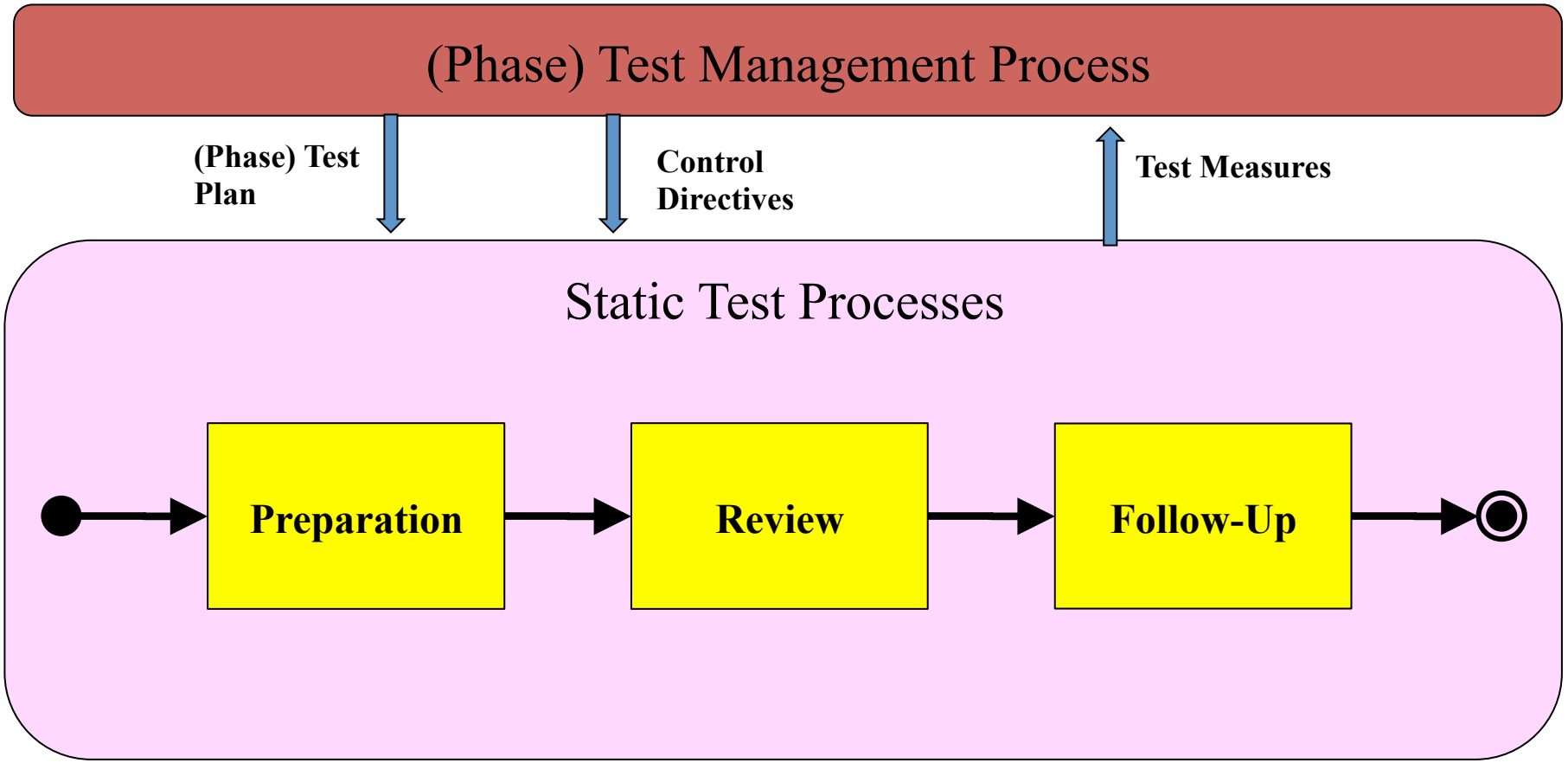
Testhanteringsprocessen



Dynamiska tester



Statiska tester

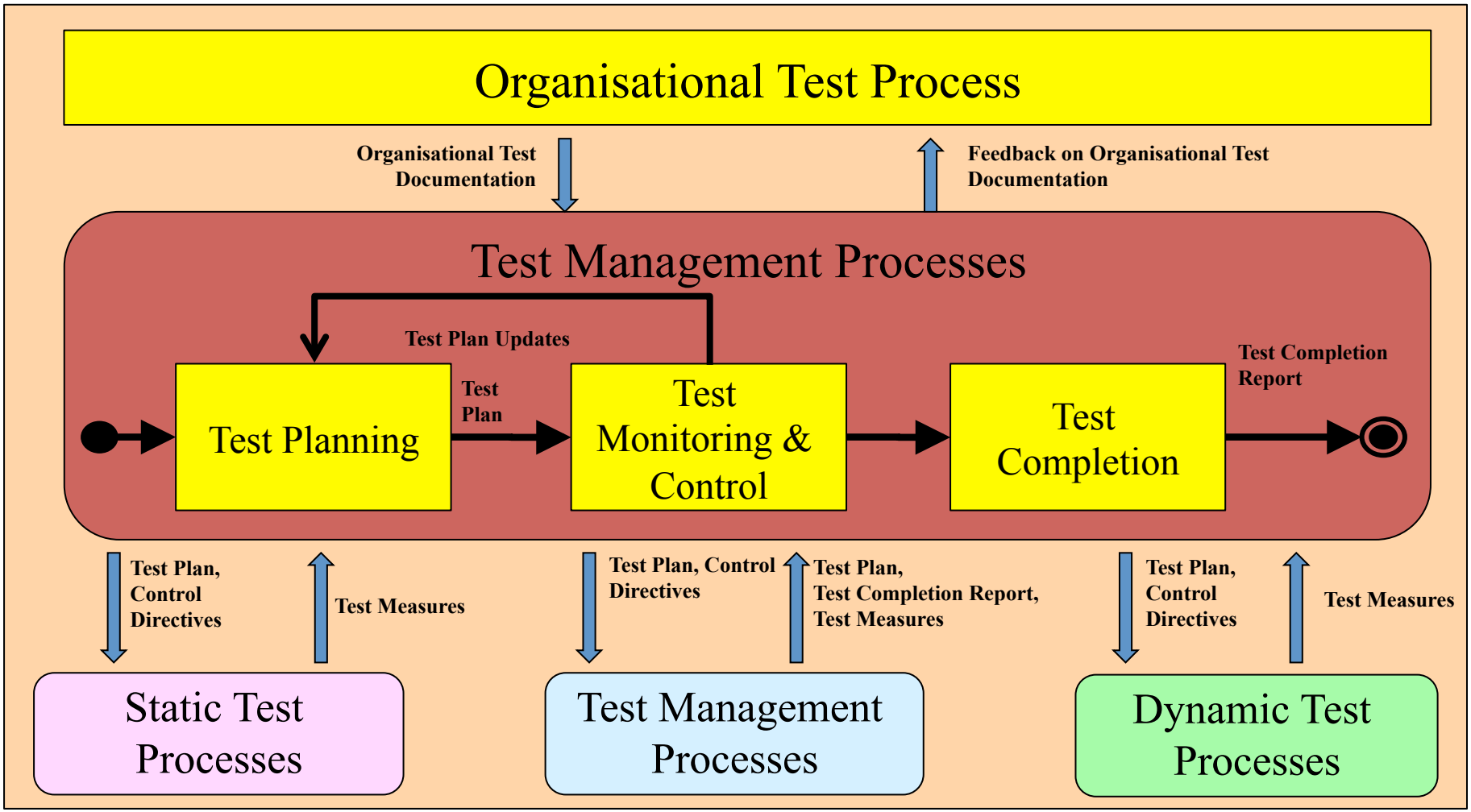


Testning

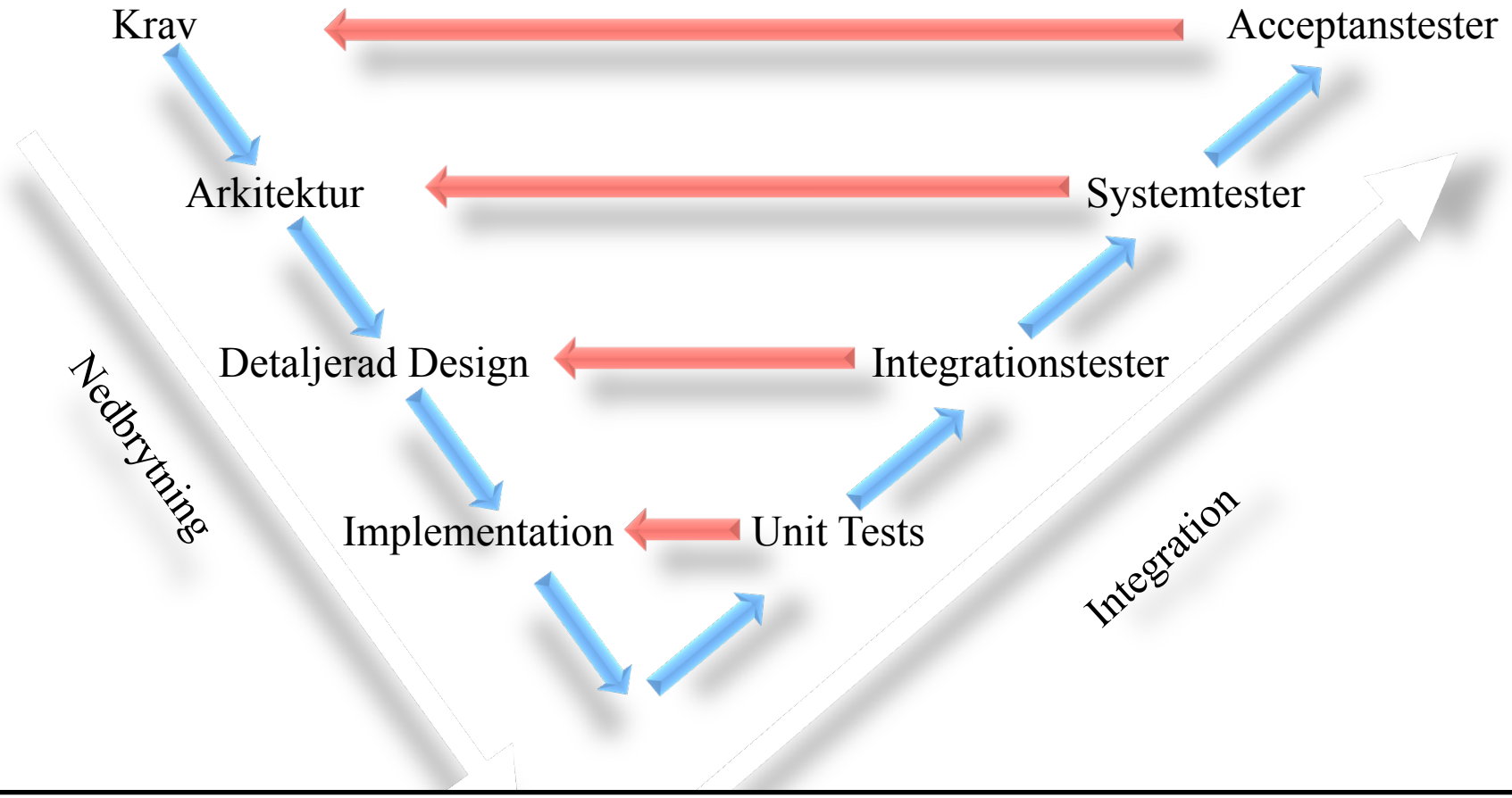
- ✓ Testning **utförs** i alla faser
- ✓ Testning kan ses som ”**destruktiv”** aktivitet – *syftet är att generera fel*
- ✓ Testning kan ses som en ”**konstruktiv”** aktivitet – *påvisa att något är korrekt*

- ✓ Testning av system ➔ Systemintegritet
  - – Testning måste ske systematiskt.
  - – Måste planeras
  - – Måste genomföras på ett strukturerat och kontrollerat sätt

Testhanteringsprocessen



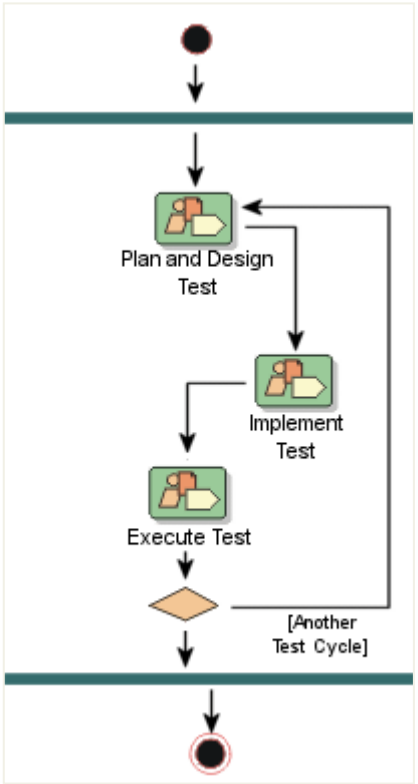
Test nivåer – V-Modellen



"Vattenfallsliknande" modell. Måste överföras till en iterativ, inkrementell, modell!

UP - Testning

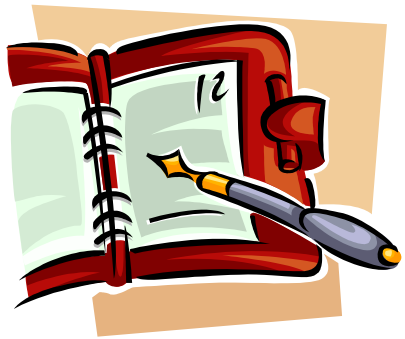
- ✓ *Testdisciplinen* fungerar som en tjänst till flera andra discipliner.
- ✓ Nyckelaktiviteter:
  - Identifiera och dokumentera kvalitetsdefekter i mjukvaran.
  - Ge rekommendationer avseende den mjukvarans kvalité.
  - Validera de *antaganden* som görs i krav, analys och design-faserna med hjälp av demonstrationer.
  - Validera mjukvarans funktioner.
  - Verifiera att kraven implementerats korrekt.



Planera testprocessen

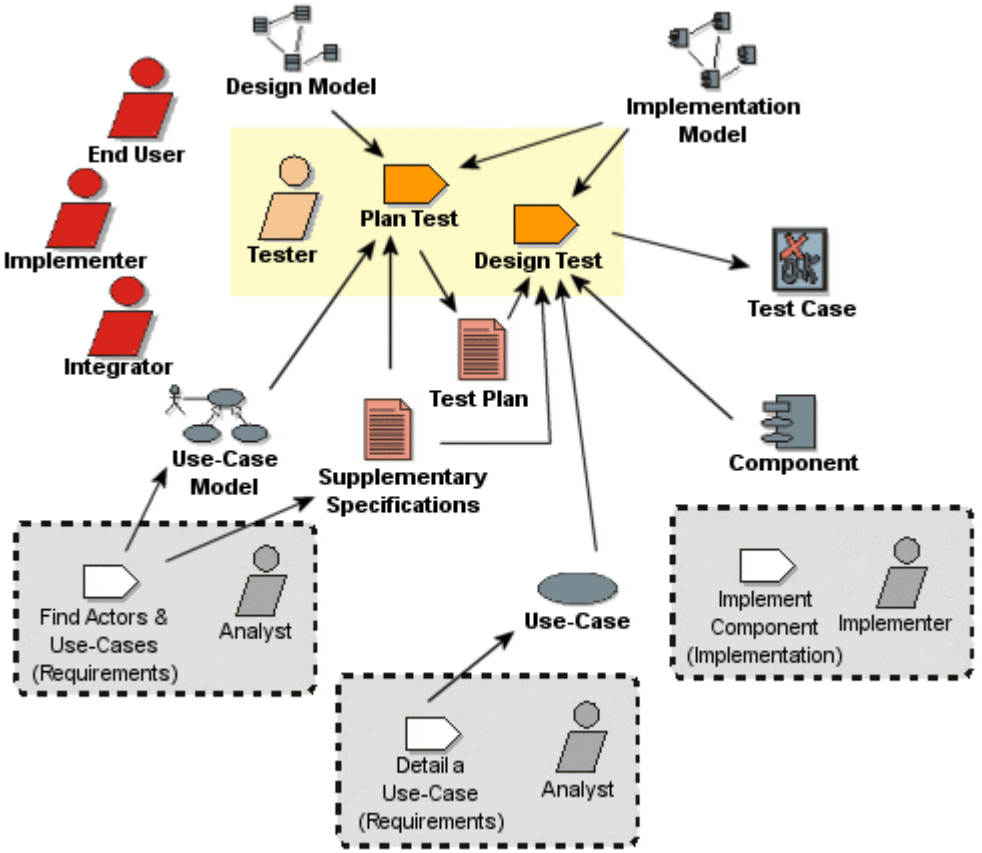
- ✓ En planering måste besvara följande frågor
  - Varför testar vi?
  - Vad behöver testas?
  - Vem skall testa
  - Hur skall vi testa?
  - Hur skall vi dokumentera?

- ✓ Planeringen kan påbörjas direkt i projektet
- ✓ Utgångspunkten är kravspecifikationen



Testplanering

- ✓ Plan
  - Identifiera testkraven
  - Riskbedömning
  - Arbeta fram en teststrategi
  - Identifiera resurser
  - Ta fram en tidsplan
  - Dokumentera i testplanen



En mall för testplaner

- ✓ Introduktion och Bakgrund
- ✓ Produktöversikt
- ✓ Testmål (objectives)
  - Funktionstestning
  - Kvalitetskrav
- ✓ Testomgivning
- ✓ Modultester
- ✓ Integrationstester
- ✓ Systemtester

  specialfall: Acceptanstest
- ✓ Resurser
- ✓ Planering
- ✓ Rapportmallar

Planering i olika nivåer

- ✓ Detta sker I flera steg, på flera nivåer.
- ✓ Jämför med projekt planering
- ✓ Testplanen ➔ Långsiktig
  - Varför testar vi, funktion & kvalitet (test objectives)
  - Vad skall testas, artefakter (test objects)
  - Testomgivning
- ✓ I varje iteration ➔ Kortsiktig
  - Planering utifrån iterationens innehåll och mål
  - Testdesign
  - Revidering av test

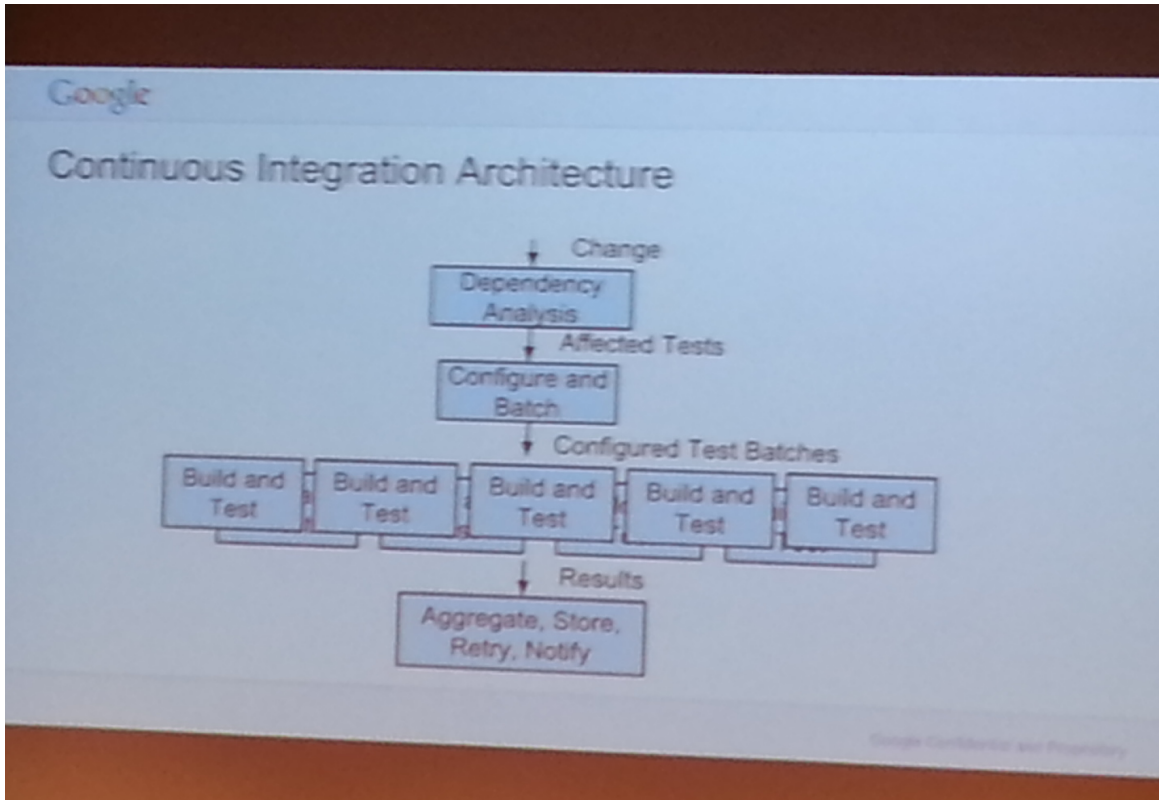
Testning kräver också utvecklingsresurser!

- ✓ Viktigt att tänka på i planeringen!
- ✓ Uppsättning av testomgivningen
  - – Testhanteringsverktyg
    - Hantera testfall, specifikation & dokumentation
    - Rapporthantering
  - – Drivers - Stubs
  - – Emulatorer/simulatorer/hårdvara i test-loopen
  - – Kontinuerlig drift och underhåll.

- ✓ Develop test-scripts
  - – Test-data and expected output

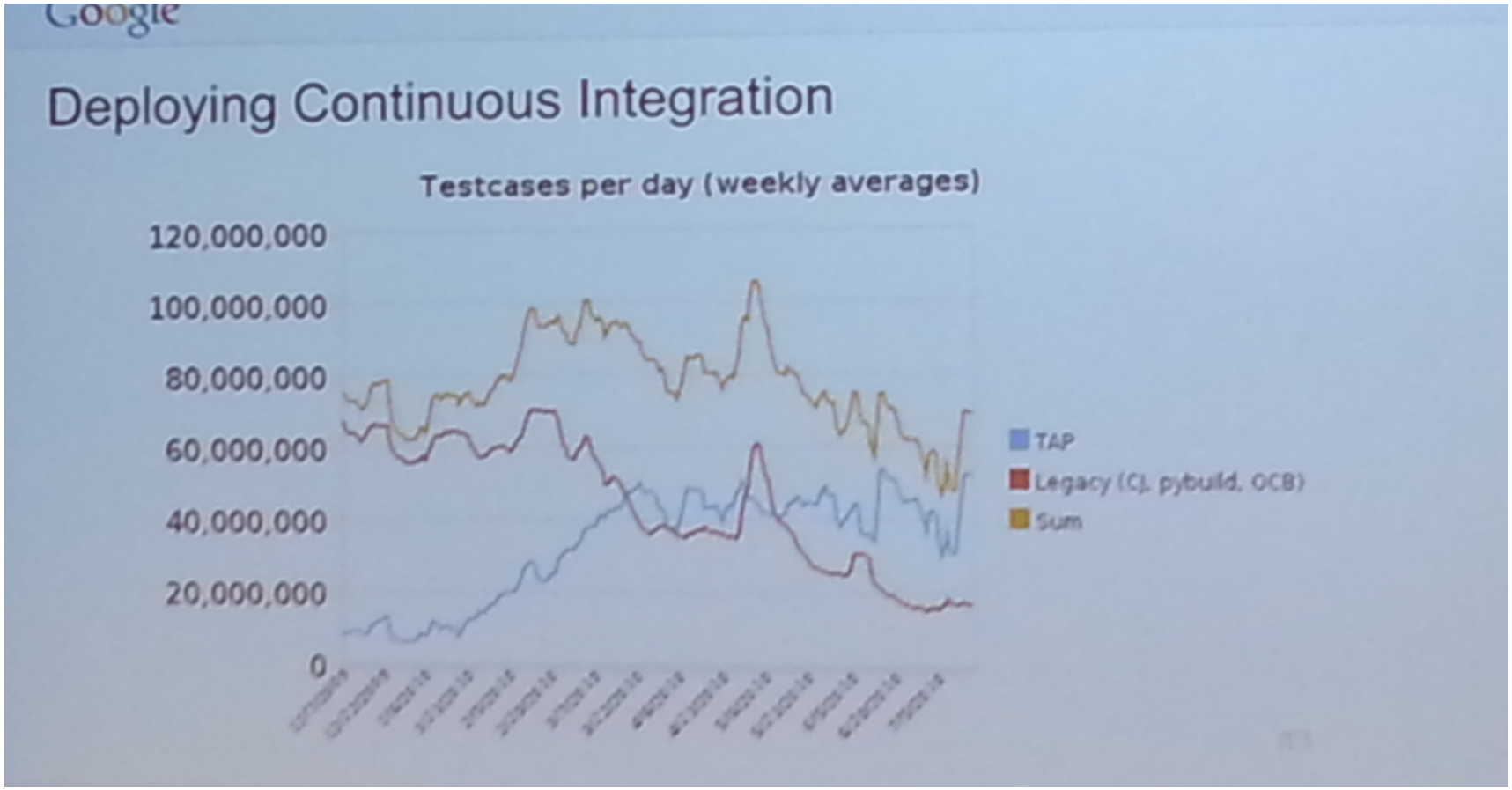
Exempel. “Den stora sökmotorn”

Continuous integration (CI), en strategi inom mjukvaruteknik där man för samman alla utvecklares arbetskopior i en delad “mainline” flera gånger om dagen.



Forts.

Visar antalet testfall som exekveras per dag (genomsnitt). Blå kurva

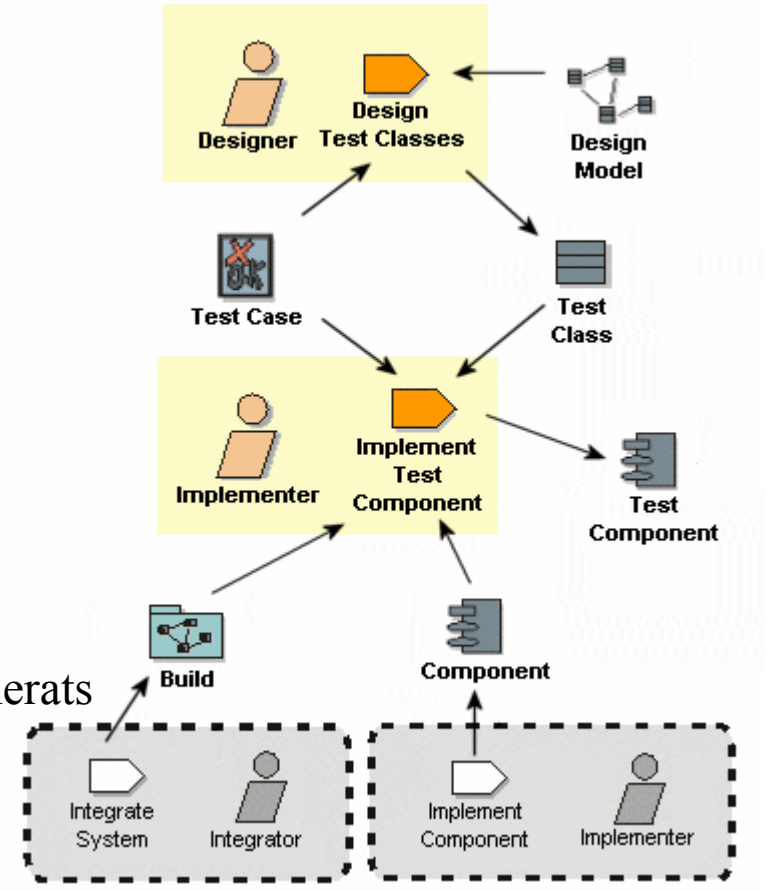


Testning i en iteration

- ✓ I varje iteration behöver man planera för
  - Förberedelser
  - Utförande
  - Analys
- ✓ Varning! Dessa aktiviteter är tidsödande!!!

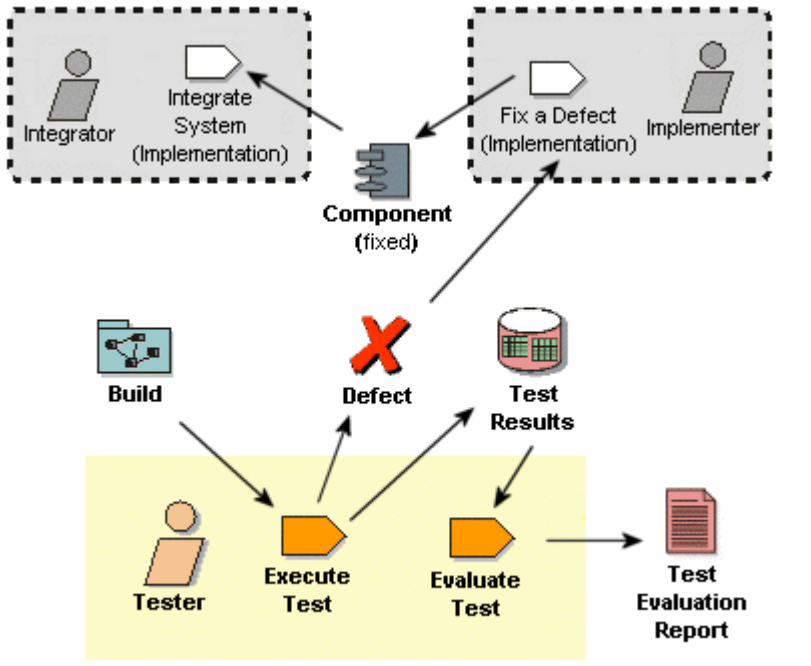
Testförberedelser

- ✓ Information från
  - Testgruppen
  - Arkitekter
  - Utvecklare/integratörer
- ✓ Vad skall göras
  - Design och implementation av testskript
  - För exekvering av testfallen i iterationen.
  - Kopplat till de releaser som planerats för iterationen
  - Glöm inte Testdata!!!



Testexekvering

- ✓ Exekvera testskripten
- ✓ Verifiera testresultaten
- ✓ Eventuellt genomföra en första felanalys för att
  - Identifiera orsak
  - Lokalisera
- ✓ Skriva felrapport
  - Feltyp
  - Var inträffade det.
  - Hur yttrade det sig
  - Koppla till
    - Testfall
    - Testskript
    - Release



Testanalys

- ✓ Syftet är att förbättra testningen
- ✓ Analysera testdata!
- ✓ Kan testerna, testdata och testskript förbättras?
- ✓ Statisktestning, exempelvis för att se täckningsgrad av tester på kod.
- ✓ Förbättra testerna

Varför?

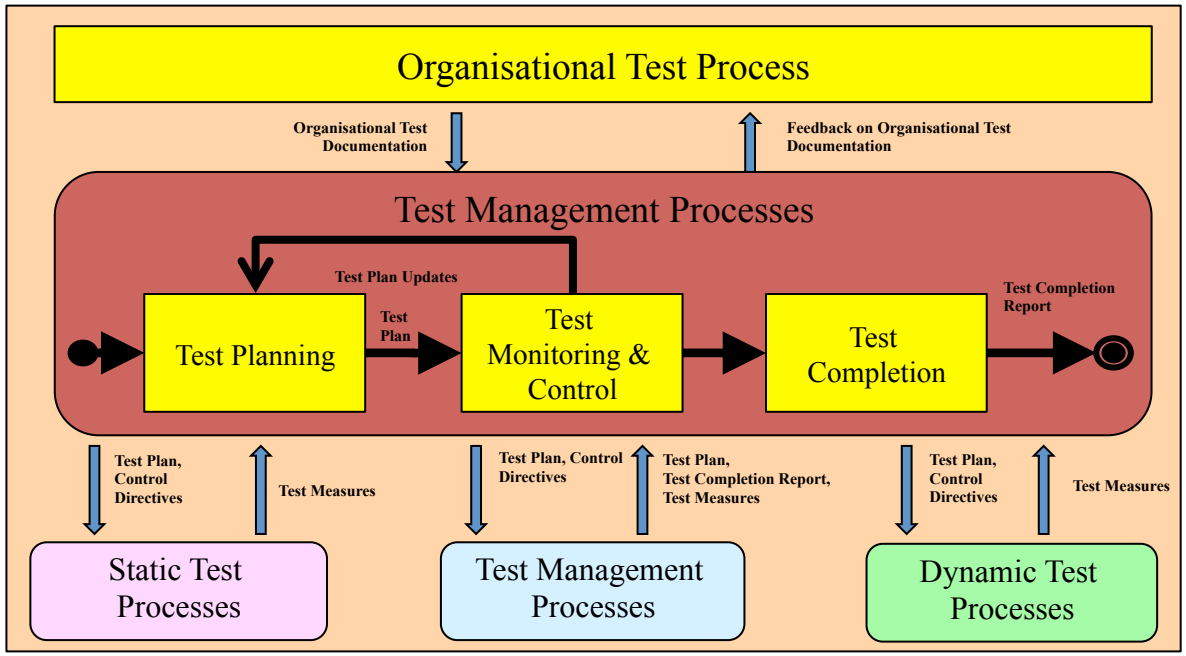
Vad?

Hur?

Test Object, Objectives & Techniques

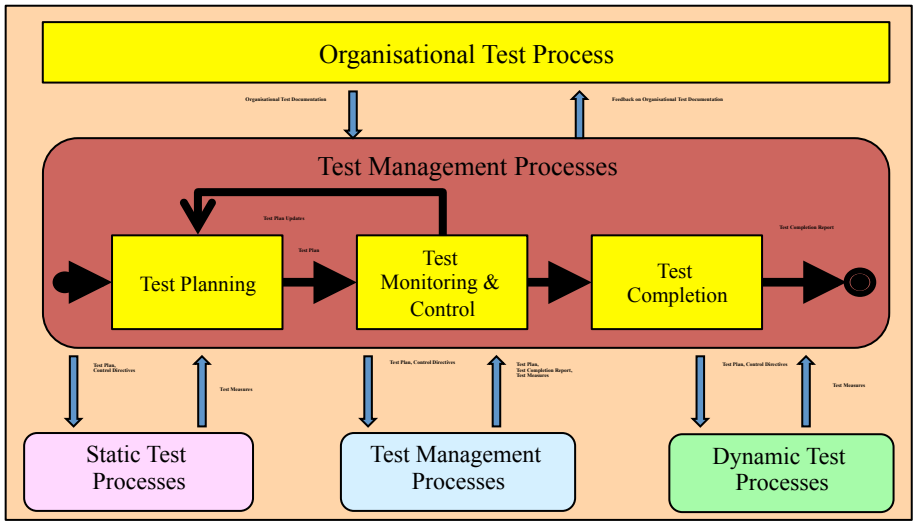
1. *Identifiera* - Test Object ➔ Vad!
2. *Definera* - Test Objective ➔Varför!
3. *Välj* - Test Technique ➔Hur!

Test Suite

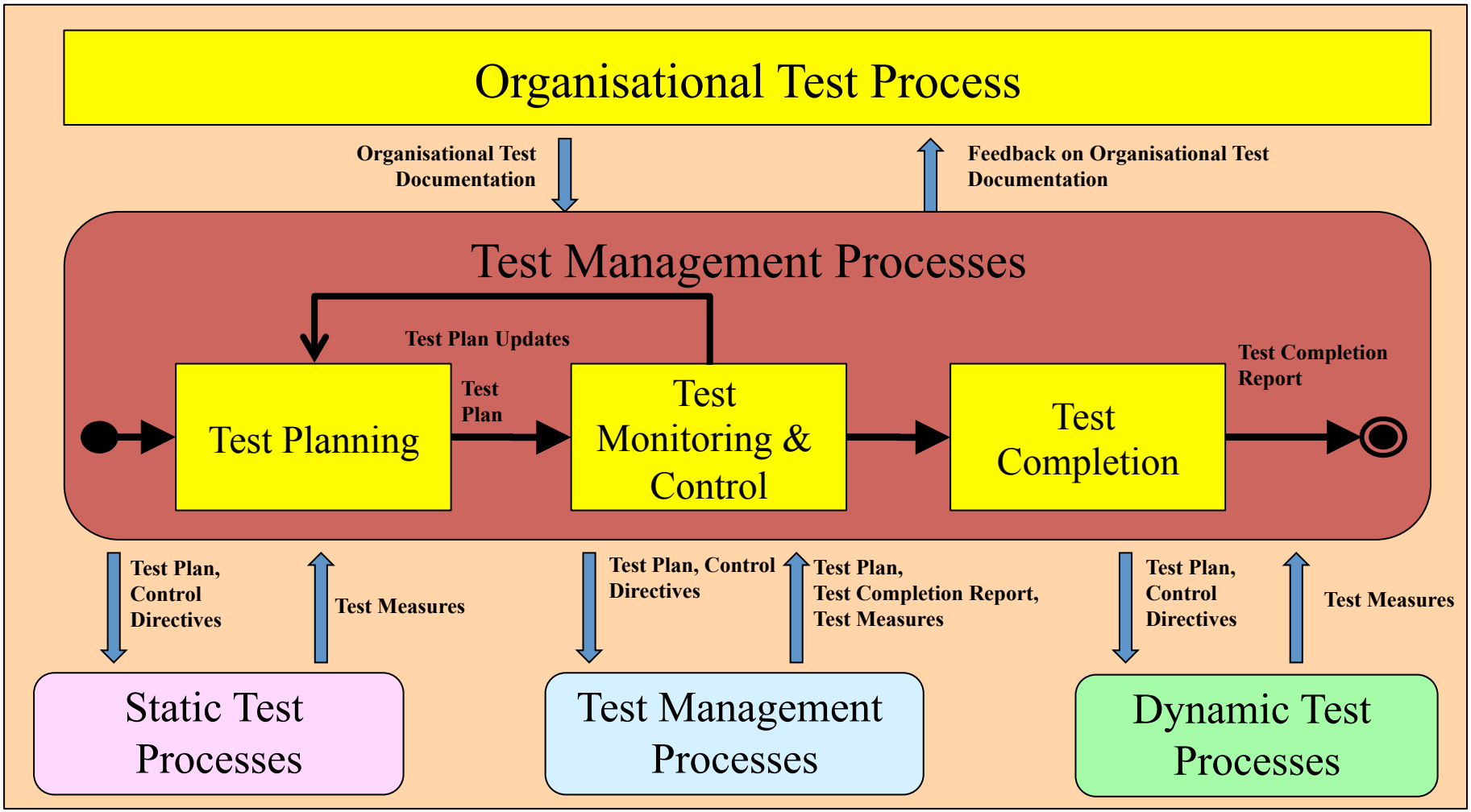


Test Suite – “Testsvit”

- ✓ 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

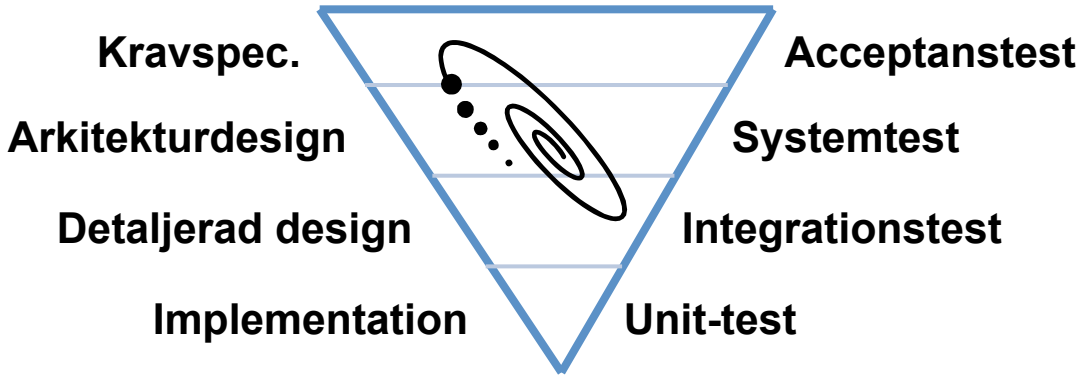


Test Management Processes



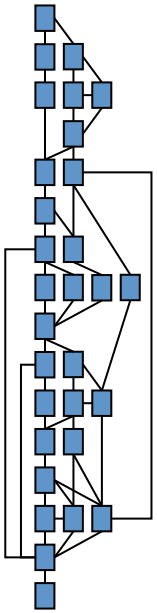
Test nivåer kopplat till faser

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

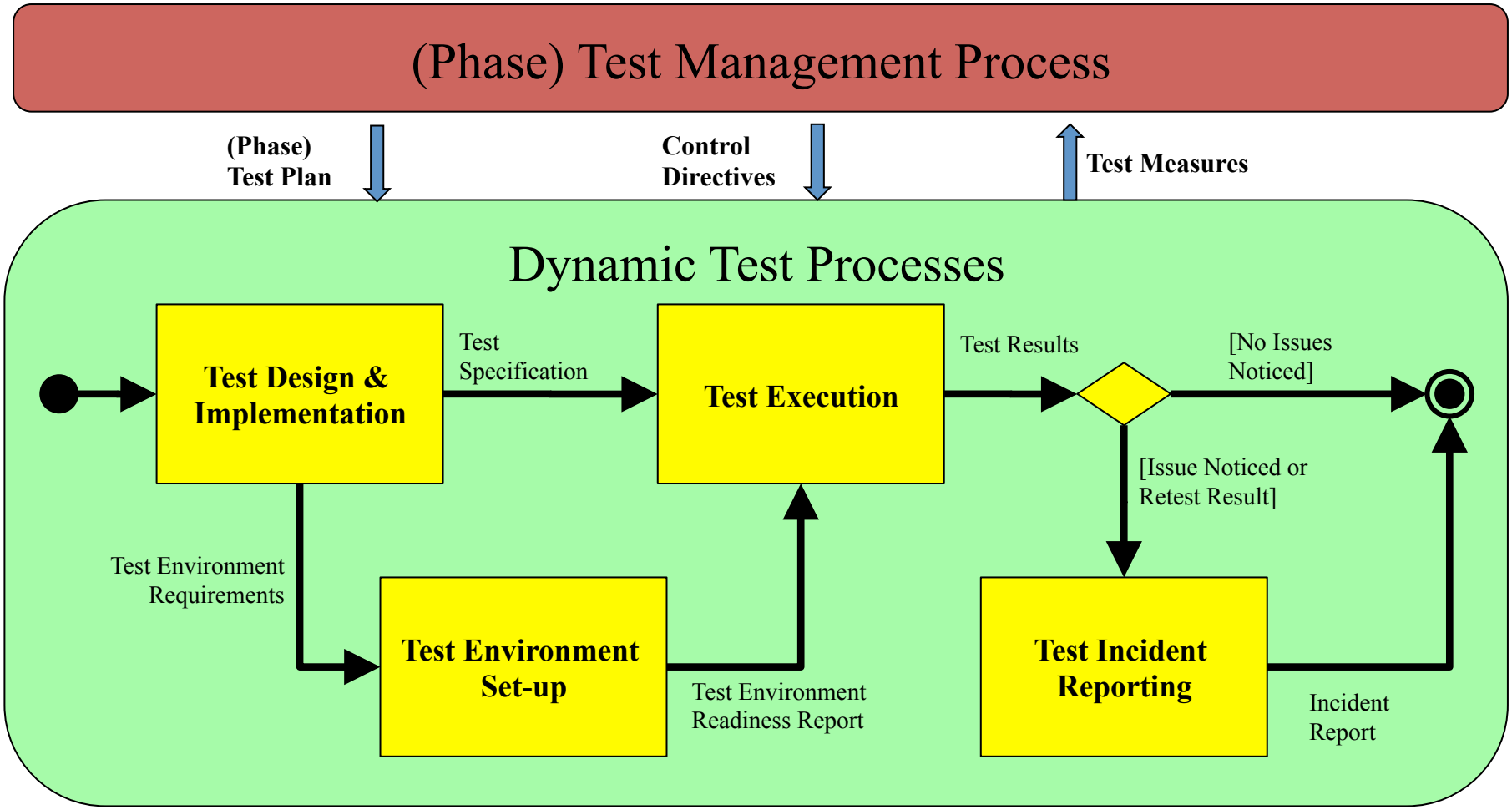


Unittester (UT)

- ✓ 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 s.k. stukturtestning (”white box testing”).
  - Utförs därför oftast av utvecklare med direkt tillgång till koden
  - Man mäter ”kodtäckning” (*code coverage)*
- ✓ Statisktestning
  - Granskningar
  - Kodstandard

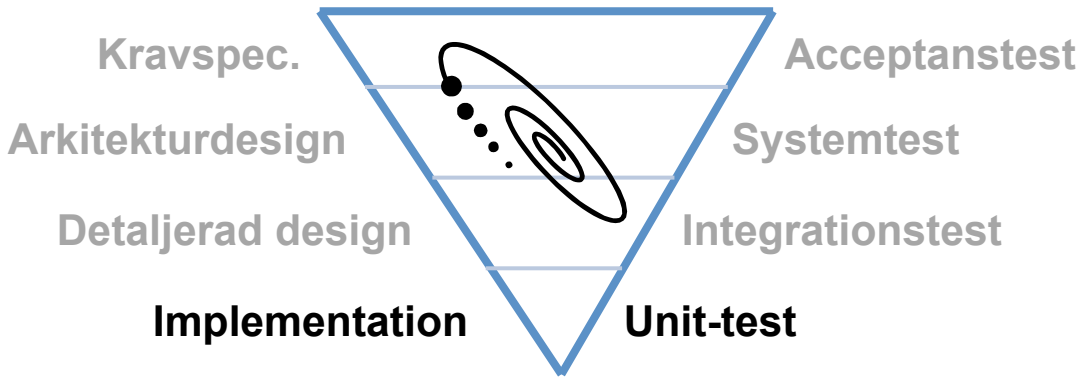


En process för dynamiska tester.



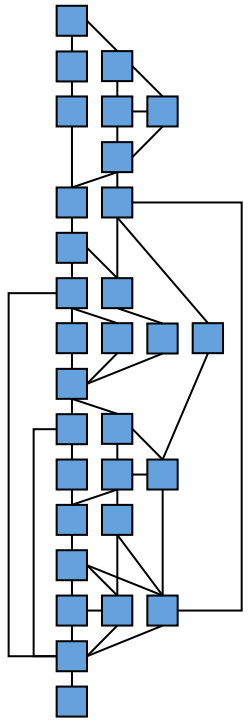
Enhetstester (Unit Test)

- ✓ För enhetstester kan man använda olika typer av verktyg för att effektivisera. Exempelvis
  - Testfallsgeneratorer och Exekveringsstöd
  - Kodtäckningsverktyg
  - Emulatorer.
- ✓ Testspecifikationerna baseras på den detaljerade designen för en enhet!

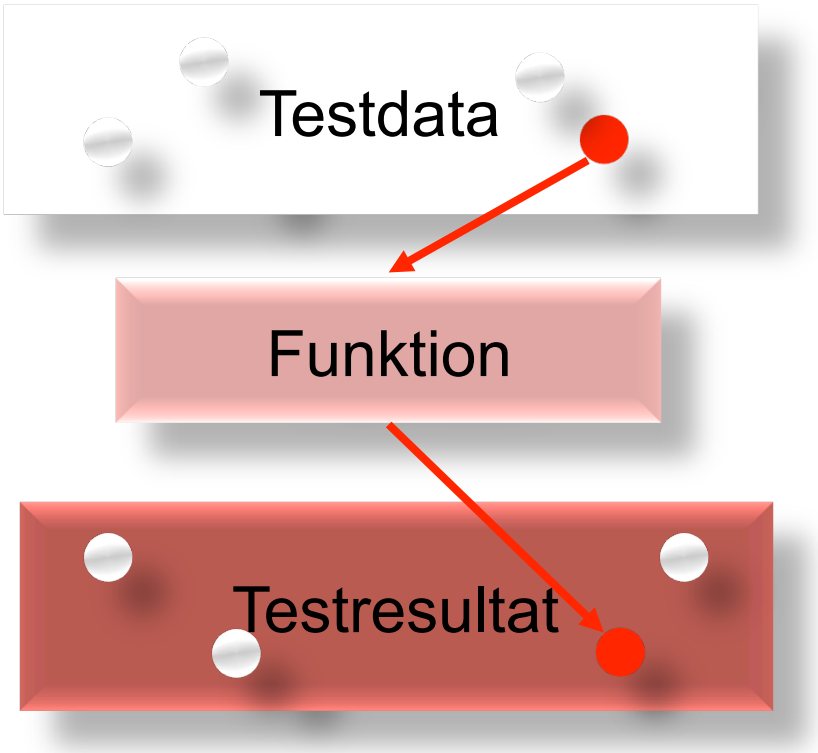


Enhetstestning, Olika strategier

- ✓ Mål (objective) – Identifiera defekter
- ✓ Utmaning
  - Testa så mycket som möjligt av koden
  - Med ett så litet antal testfall som möjligt
  - Optimeringsproblem!!
- ✓ Strategier
  - Black-box, enheten ses som en funktion.
  - White-box, strukturtestning med kodtäckning.



Strategi – Black-box Testning



Indata ”mappas” till förväntat ”resultat”

Black-box testning – Category Partitioning

- ✓ Delar upp *indatadomänen* i *kategorier* som sedan delas upp i *val*

- ✓ Analysera enheten som skall testas (endast signatur och specifikation)
- ✓ Identifiera *partitioner* (uppdelningar)av *ekvivalenta indata* och *utdata*.
- ✓ Tar bort redundans och minimerar storleken på testsviten

- ✓ "Tittar på" gränsvärden!
- ✓ Testaren känner inte till hur enheten ser ut internt.
- ✓ Tekniken är en systematisering av hur erfarna testare arbetar.

Category Partitioning, steg

- ✓ Bryt ned den funktionella specifikationen i enheter.
- ✓ Identifiera parametrar och omgivningsvillkor
- ✓ Identifiera kategorier i informationen.
- ✓ Partitionera vare kategori i val.
- ✓ Skriv testspecifikationer för varje enhet

Sortering exempel

Exempel: **Sortering**
Specifikation:
Input: array av varierande längd och av godtycklig typ
Output: Permutation av indata, sorterad
Minimumvärde
Maximumvärde
Parameterar: Array

- ✓ Kategorier:
  - Array's size
  - Elementtyp
  - Maxvärde
  - Minvärde
  - Position för max och minvärden
- ✓ Val:
  - Längd: { 0, 1, 2 .. 100, 100 .. INF }
  - Typ: { Integer, Character, Array, Record, … }
  - Max: . . .

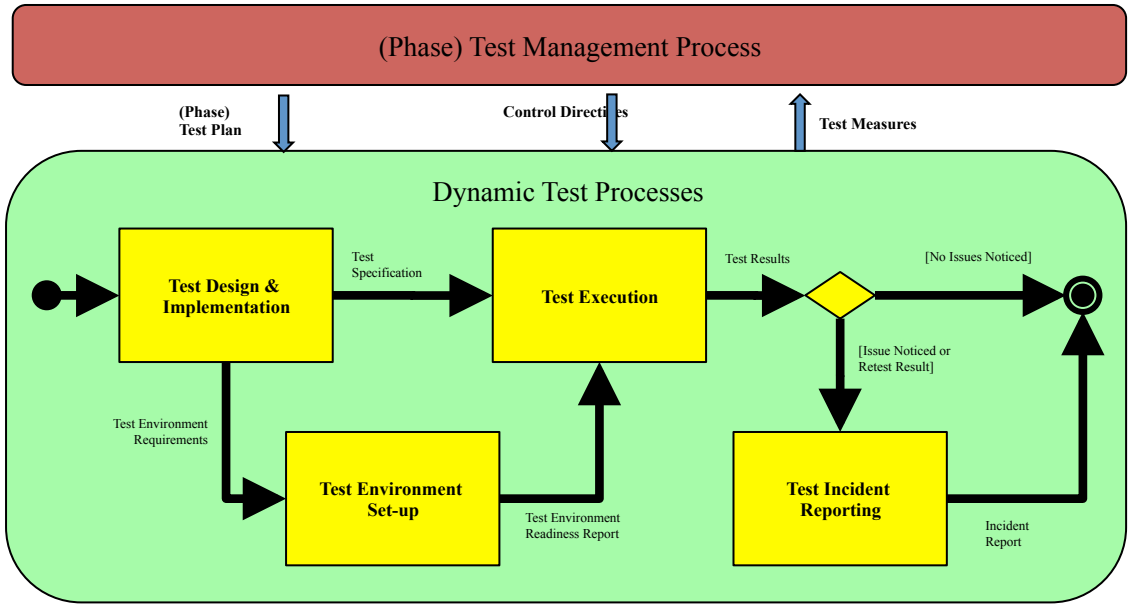
White-box testning (strukturtestning)

- ✓ Analysera enhetens
  - Signatur,
  - Specifikation,
  - Implementation
- ✓ Testfallen arbetas fram efter implementationens struktur (call-graph)
- ✓ Generera test-data så att så mycket som möjligt av koden ”täcks”

- ✓ Hur många testfall behövs?
- ✓ **Coverage criteria**

White-box testning– Process

1. Välj *coverage criteria*
2. Generera *Control flow* grafer
3. Instrumentera koden för att “mäta” täckning
4. Arbeta fram testfall
5. Exekvera testerna
6. Analysera “täckning”
    1. Upprepa 4
    2. Avsluta



Inför exekvering, förberedelser

- ✓ Vad kan behövas
- ✓ Stubs, simulerar funktionalitet som ännu inte finns
- ✓ Driver, “ersätter” anropande metod som inte heller implementerats
- ✓ Sequencing, metoder som sätter upp och tar ned testomgivningen
  - Exempelvis, initierar en databas innan testerna
  - Återställer den efter testet är avslutat

Iterativutveckling - regressionstester

- ✓ I varje iteration sker ett inkrement, d.v.s. ett “tillägg”

- ✓ Tyvärr räcker det inte att bara testa “tillägget”.
- ✓ Alla beroenden måste också testas.

- ✓ Att testa sådant som inte borde ha förändrats i innevarande iteration kallas regressionstestning.

- ✓ Exempel

Nästa vecka

- ✓ Testning i praktiken
- ✓ Test driven design

- ✓ SCRUM