**Frågor runt testning ur ISTQB-perspektiv. (Maxpoäng 23p)**  
  
Varför testar vi mjukvara och dokument före under och efter utvecklingsprocessen? 3p

Varför vi testar mjukvara före, under och efter utvecklingsprocessen är på grund av att kunna hitta fel tidigare och att få färre buggar. Dessutom så är det lättare för testare och utvecklare att föra en bra kommunikation sinsemellan, när de arbetar parallellt. Mindre risk för missförstånd uppstår.

Det testas mer i agil utveckling (kontinuerlig testning under utvecklingsprocessen), iteration och regression är två nyckelord i sådan testmiljö.

Utvecklare komponerar nu allt oftare testcase innan de har skrivit koden i mjukvaran. Det gör att de måste tänka på många olika aspekter som rör den nya funktionaliteten innan kodningen påbörjas.

Den kontinuerliga testning som sker i en agil utveckling är mer effektiv och antalet buggar minskar. Det är svårare att rätta till fel i den klassiska vattenfallsmodellen, då det blir stora testblock som gör det svårare att spåra ”vem”, ”vad” och ”hur”.

Idag så är kraven om kvalitet på mjukvara väldigt hög och eftersom tolerans mot buggar är låg, så krävs kontinuerliga tester under hela utvecklingsprocessen. Ju tidigare en bugg hittas, desto mer kostnadseffektivt blir det.

## Vad skiljer Ekvivalensklass partitionering från Gränsvärdeanalys? 3p

Gränsvärdesanalys och Ekvivalenspartitionering är båda testfallsstrategier i Black-Box testning.

Partitionering av ekvivalens: Denna metod används vanligtvis för att minska det totala antalet testfall s till en begränsad uppsättning testbara testfall, som fortfarande täcker maximala krav. Kort sagt är det processen att ta alla möjliga testfall och placera dem i klasser. Ett testvärde väljs från varje klass under testningen.

Gränsvärdesanalys : Handlar om att testa de yttre gränserna av ekvivalensklasserna. Ex. Du har en Ekvivalesnklass som testar belopp för kontoöverföring på bank. Du ska kunna lägga in max belopp som en kontoöverföring på bankkonto, du skriver in maxbeloppet och detta skall fungera. Ibland så missar programmerare enkel logik (ex. ”belopp <= 20 mkr”)

Vad menas med Exhaustive testing vid testning av mjukvara? 1p

Det är en process där du testar precis allt som går att testa, bara för att vara helt säker på att produkten inte kan krascha. Det testas alltså alla möjliga kombinationer av inputs, scenarier, och situationer som kan uppstå. Om det finns en stor budget och mycket tid, så testas allt. Detta är nästan omöjligt.

Är detta ett rimligt mål vid utveckling av majoriteten av mjukvaror? Varför, varför inte? 2p

Det skulle givetvis vara önskvärt att all programvara testades på detta sätt. Då det blir ett väldigt robust och säkert program. Dock tar det alldeles för mycket tid och energi att testa programvara på detta sätt. Det är dessutom väldigt svårt att finna alla möjliga testscenarion. Det är också väldigt svårt för ett program att klara av samtliga tester. Om det skulle vara rimligt att använda sig av Exhaustive testning beror helt på programmets storlek, men i de flesta fall så är denna testprocess både dyr och tar otroligt mycket tid.

## Varför kan/kan inte test av ekvivalensklasser och gränsvärdesanalys ersätta varandra i mjukvarutestning? 2p

Det är allmänt känt att inmatningsvärdena vid de yttersta gränserna orsakar fler fel i systemet. Därför är det viktigt att man använder sig av gränsvärdesanalyser i sina tester. Ekvivalensklasser täcker oftast maximala krav av testbara fall. Dessa två testmetoder jobbar i symbios vad gäller black-box testning och kan därför inte ersätta varandra.

Kan testning bevisa att mjukvaran är felfri? Varför/Varför inte? 2P  
Det viktiga när ett program testas är att det finns en tydlig specifikation över vad programmet skall kunna göra. Det är alltså viktigt att prioritera testningen baserat på specifikationen över funktionalitet. Gör programmet det som det är tänkt att göra? Testning kan endast påvisa förekomsten av fel i ett program, aldrig frånvaron av fel. Det finns alltså inga garantier för ett felfritt program för att det är testat. Det är viktigt att man väljer sin testdata med stor omsorg för att kunna fånga så många av felen som möjligt.

Vad skiljer TDD (testdriven utveckling) från ”vanlig” unittesting? 3P

TDD= Test Driven Development. Detta är en viktig del i agil utveckling, där testfallen skrivs innan utvecklingen av programvaran. Testerna beskriver hur programmet skall fungera och därför kommer vissa funktionaliteter fram som kanske inte hade tagits med om koden skrivits först.

Unittestning är små bitar kod som testar om ex. en utav dina metoder i en klass fungerar som den skall. När du utför ”vanlig” unittestning, så skriver du först en metod och sedan skriver du olika Unittests som kollar att metoder gör vad den skall. När du använder dig av TDD, så skriver du Unittestet först och sedan skriver du precis så mycket kod du behöver för att få testet att gå igenom.

Vilka eventuella fördelar (minst 3) och vilka eventuella nackdelar finns det med TDD jämfört med vanlig unittestning? 4p

Fördelar med TDD jämfört med vanligt unittestning:

Du kodar mot ett resultat mer effektivt. Eftersom du skriver testerna först, så får du en tydlig bild av hur koden i metoden skall se ut. Detta gör det mer tidseffektivt och du behöver eventuellt inte göra om din metod, som du kanske behövt om du kodat först. Om du skriver testerna först, så håller sig funktionen i metoden till det som metoden skall göra. Annars kan det lätt bli att metodens funktion svävar iväg och gör mer än vad den behöver.

Nackdelar med TDD :

Det kan bli så att utvecklingstiden för ett projekt ökar. Det kräver en hel del förarbete för att börja med testning innan kod är skriven. Det är viktigt att utvecklaren har en tydlig bild av kraven innan den börjar skriva testerna.

Varför vill vi i möjligaste mån automatisera olika typer av tester? 3p  
Mycket handlar om att det är hård konkurrens och en snabbrörlig värld som gör att allt måste gå fortare men samtidigt hålla hög kvalitet. Automatisering ger oss möjligheten att snabbt få feedback på ett systems status om vi råkat ha sönder något. Maskiner utför operationer snabbare än vad vi människor kan, samtidigt som de kan utföra tusentals simulationer i en snabb takt. Automatiseringen gör att vi kan utnyttja tiden på ett bättre sätt och testare kan lägga sin tid på djupare insikt om nya områden, istället för att lägga tid på tidskrävande regressionstester. Testautomatisering blir snarare ett viktigt stöd och komplement till den manuella testningen och ett bra verktyg till bättre kvalitet.