**Svar:**

1. Testning försäkrar att all kod möter kvalitetsstandarden innan den skickas iväg/används. Detta försäkrar en stabil miljö där kvalitet är nyckeln.   
   Under utvecklingstiden så sparar man både tid och pengar. Samt hjälper utvecklare skriva bättre och snyggare kod. Detta genom att testa kontinuerligt före, innan och efter utvecklingsprocessen.
2. Ekvivalensklasser delar upp set av testklasser att testa.   
   Medans gränsvärdesanalys testar mellan testklassernas gränser.
3. Exhaustive testing är en typ av mjukvarutestning där alla möjliga data kombinationer är använda för testning. Det är en typ av försäkring, då man testar alla scenarios och data. Exhaustive testing är en process som försäkrar att produkten inte kraschar eller förstörs av slumpmässiga anledningar.
4. Detta är dock inte rimligt vid majoriteten av utvecklingen av mjukvaror. Då det tar alldeles för lång tid och för mycket resurser och pengar. Dock så är det nog det ideala scenariot vid utvecklingen då man kan försäkra sig om en väl testad produkt.
5. Ekvivalensklasser och gränsvärdeanalyser jobbar i symbios på större projekt. För att kunna dela upp och tackla såväl black box testing samt högintensiva projekt som har många kalkyleringar.  
     
   Detta innebär att man kräver båda, på många projekt, för att kunna dela upp projektets tester med hjälp av ekvivalensklasser och lyckas testa gränsvärdena.
6. Testning kan inte bevisa att mjukvaran är felfri. Enbart eliminera mängden fel. Det är omöjligt att förutse alla fel som kan inträffa. Man kan därför inte säga att ens mjukvara är felfri, utan bara väl testad.
7. Unittesting är att skriva små tester som enskilt testar en liten del, funktion eller beteende av objektet. TDD är istället en tankeprocess som resulterar i unittester. “Tänka i testväg” tenderar att ge mer välgenomtänkta och omfattande tester. Och en enklare design att vidareutveckla mjukvaran.
8. Fördelar:   
   För att du skriver små tester, en i taget, så tvingar det att ens kod blir mer modulär och välstrukturerad. Det gör det också lättare för en att lära sig grunderna för en god design.  
     
   Detta innebär att strukturen på mjukvaran är bättre och problem som hade kommit upp utan TDD elimineras tidigt i processen.  
     
   Det gör också att koden är lättare att underhålla och ändra. Eftersom man har en så bra grundstruktur så är det som ett säkerhetsnät när man väl vill ändra något i koden.   
     
   Nackdelar:  
   TDD kan sakta ner utvecklingsprocessen avsevärt, speciellt när man är ny till sättet att arbeta på. Det innebär också att alla i utvecklingsteamet måste vara ombord med sättet man arbetar på och förstå vikten av det.  
     
   Det kan också vara svårt att anpassa o anamma TDD till redan befintliga projekt och svårt att skriva koden. Ännu svårare att skriva koden väl.
9. Testautomatisering är det bästa sättet att öka effektiviteten och den övergripande rapporteringen av mjukvarutestning. När de automatiserade testerna väl är skapade och implementerade så kan dem enkelt bli upprepade och vidareutvecklade att utföra ytterligare uppgifter som tidigare var omöjliga med manuella tester.