Jesper Eriksson, MUT19

1. Vi testar mjukvara för före under och efter utvecklingsproccessen eftersom att man då får så stor genomgång av programet som möjlig. Man gör tester före för att kunna enklare göra metoder som kan konstrueras via testerna, man testar under för att det hela tiden kan komma nya metoder och man gör det efter så att man kan vara hyffsat säker på att det mesta ska fungerar som det ska. Och man testar efter så man kan säkerställa att allt fungerar som det ska.
2. Ekvivalensklass partionering skilljer sig från Gränsvärdeanalys på det sättet att i ekvivalensklass partionering har man tre värden att testa på ett incorrect värde i botten av ett värde intervall och ett i toppen av värde intervaller (Exempel intervallet 18 – 60 borde då testas 17 – 25 – 61 så att man har ett correct värde och två incorrecta) Där 17 ska ge ett resultat 25 ger ett annat resultat och 61 ger ett tredje resultat.

Medans i gränsvärde analys använder man sig av ekvivalenspartionerna men då har man istället 4 – 9 värden att testa på (Exempel i samma intervall så testar man om det är en full gränsvärde analys så testas 17,18,19 24,25,26 och 59,60,61)

1. Exhaustive testing är när man testar alla möjliga scenarior och use cases. Man testar alla möjliga kombinationer av inputs i en metod för att se så att den är helt fungerande. Man testar helt enkel ALLT som går att testa i varenda metod.
2. Nej, Exhaustive testing är inte ett rimligt mål vid utvecklings processen eftersom att det tar väldigt lång tid att skapa alla testerna och det skulle kosta väldigt mycket pengar för företaget om man skulle skriva tester via exhaustive testing.
3. Man kan inte ersätta Exhaustive testing med Ekvivalensklass partionering eller gränsvärdeanalys eftersom att exhaustive testing inte är ett rimligt mål att uppnå i en utvecklingsprocess, Det är ett mycket bättre tillgångssätt att använde sig av EkvivalensPartionering och gränsvärdeanalys för att bygga upp sina tester och få dem så bra som möjligt.
4. Testning kan inte bevisa att mjukvaran är helt felfri eftersom att det finns så extremt många fall som skulle kunna göra att ett program skulle antingen krasha eller göra något annat fel. Man kan komma väldigt nära men det finns alltid den humana faktorn som gör att det kan uppstå fel man inte tänkt på tidigare.
5. Skillnaden på Testdriven utveckling och “vanligt UnitTestning är ganska stor, Under en “vanlig” unittestning så testar man codens moduler (units) på en egen basis. Testerna kan vara manuala eller automatiserade och kan skapas innan koden är skriven men också efter koden är skriven. Testdriven utveckling däremot där skriver man unittester för att få ihop en utvecklings ram. Man skapar oftast automatiserade tester i början av utvecklingen som skal faila och sedan ska man skriva kod så att dessa tester inte failar längre. Ingen kod får skrivas innan dem test som man har har failat och man är nöjd med upplägget av failande tester.
6. Att börja utvecklingen med att skriva test säkerstället att den kod som skrivs blir godkänt testad. Sedan att veta att kod man skriver som fungerar mitt under utvecklingen av koden är också positivt då man inte måste testa programet efter varje kod man skriver. Det bli ett enklare flöde igenom utvecklingen. Det är även positivs att kunna säkerstella funktionaliteten hos en funktion i taget så att man inte “tror” att man är klar och så kommer det upp massa problem i framtiden i utvecklingsprocessen. TDD gör även utvecklings processen snabbare och minskar tiden som går till utvecklingen.

Nackdelar med TDD är att viss funktionalitet blir svårt att testa, och då blir det väldigt svårt att skriva testerna innan utvecklingen av programmet har börjat. TDD kräver också att hela utvecklings teamet tar en gemensam beslutning att man ska arbeta med TDD för om det är någon i teamet som inte utvecklar via TDD kan det uppstå en väldig massa problem. TDD kan även ses som negativt om man har mycket legacy-kod, vilket kan ses som en negativ aspekt i TDD utveckling.

1. Vi vill automatisera olika typer av tester för att kunna göra testningen enklare och göra så att det inte tar så lång tid att testa programmet. När man automatiserar test blir det mycket säkrare att göra ändringar i koden då om man gör något som får programmet att inte fungera längre så får man det svart på vitt direkt när man går igenom testningen. Automatiserade tester kan även funka som documentation över koden och dess funktionalitet då man kan se allt som ska fungera.