Laboration 2 – Grundläggande begrepp och principer

Författad av: *Gabriel Afram, Thomas Astner, Alex Darborg, Andreas Edin, Adam Hjernquist, Joakim Sundqvist* och *Mattias Vängman*

Kravspecifikation (SRS)

En kravspecifikation är en beskrivning av det mjukvarusystem som utvecklas, vilket arbetas fram av både kund och entreprenör samt fastställer att båda parter är överens om hur slutprodukten ska se ut. Det planlägger funktionella- och icke-funktionella krav samt fastställer vilka typer av användarinteraktioner som mjukvaran måste stödja.

Språket i ett SRS-dokument är skrivet på ett sådant sätt att kunden ska kunna förstå vad som står utan vidare teknisk expertis, vilket innefattar både text och analytiska modeller såsom UML (ex. Use Case Diagram) och ER-Diagram.

Kravspecifikationen låter entreprenören estimera det ungefärliga priset och hur lång tid ett projekt kommer att kräva samtidigt som det fungerar som ett hanteringssystem för vad kunder har att förvänta sig. Om kundens krav förändras under utvecklingsprocessen kan priset och tiden som krävs att kunna beräknas på nytt utifrån dem nya förutsättningarna.

Kravfunktionalitet

Ett **funktionellt krav** är vad som definierar ett systems funktion eller funktionen av en komponent i systemet; hur det krav som kunden ställer i kravspecifikationen faktiskt fungerar – vad det **gör**. Dessa kan exempelvis beskrivas med hjälp av Use Case diagram och/eller textuella beskrivningar. Ett **icke-funktionellt krav** används i konjunktion med funktionella krav för att beskriva hur systemet i stort ska fungera, såsom prestanda-, underhålls-, migrations- och användbarhetskrav.

Ett funktionellt krav definierar vad ett system ska göra medan ett icke-funktionellt krav definierar hur systemet ska vara. Ett funktionellt krav utrycks som specifika funktioner såsom exempelvis ”systemet skall beräkna [någonting]” eller ”systemet ska hämta information från [databas]”. Ett icke-funktionellt krav kan utryckas som ”systemet skall vara säkert” eller ”systemet skall vara stabilt”, osv, i en mer ospecificerad ton.

Prototyp

En prototyp inom mjukvaru-utveckling är en ofärdig version av ett system under utveckling. Prototypen används under utvecklingens gång som test-system för att försöka visualisera det färdiga systemet/projektet och förbättra kommunikation mellan deltagare i arbetet genom att ge en gemensam bild av slutprodukten samt vart i arbetsprocessen man befinner sig.

Single Responsibly Principle

En funktion, klass eller modul ska bara förändras/modifieras på grund av **en** (och endast **en**) anledning. Denna princip används för att underlätta underhåll och organisering av system samt snabba upp/bibehålla utvecklingstakten.

Anta att ett program läser in en fil och gör matematiska kalkyler baserade på informationen i filen. Här kan exempelvis två händelser ge anledning till att källfilerna måste redigeras; att filformatet förändras eller att den matematiska kalkylen är ineffektiv/gammal. Dessa områden förändras alltså på grund av helt olika anledningar, d.v.s. input och prestanda. Dessa två förändringar ska enligt **SRP** vara två enskilda ansvarsområden (responsibilities) och skall därför skrivas i skilda klasser/moduler för att underlätta underhåll (uppdatering) av systemet.

Open Closed Principle

Funktioner, klasser eller moduler ska vara öppna för utvidgning/utbyggnad **utan att** ursprungskoden behöver skrivas om. Denna kraftfulla princip är direkt relaterad till objektbaserad programmering och **arv** (inheritance) där vi ska kunna bygga subklasser till redan skrivna klasser utan att huvudklassen behöver modifieras. Denna princip är även direkt relaterad till SRP eftersom OCP underlättar underhåll och vidare utveckling eftersom vi behöver göra färre förändringar vid uppdatering.

Liskov Substitution Principle

Om X är en subtyp av Y så måste instanser av objektet X ersätta objekt av typen Y – utan att förändra någon utav dem önskvärda egenskaperna hos Y. LSP handlar om valet mellan att skriva en subklass eller använda en annan strategi för att uppnå den önskvärda funktionaliteten. Med andra ord så kan vi alltså inte i programkod säga att en boll är en punkterad boll, en fyrkant är en rektangel eller att ett fordon är ett flygplan, utan tvärtom (en rektangel är en fyrkant).

Kom ihåg:

I projektrapporten beskriver vi hur ovanstående begrepp har utnyttjats i själva projektet med tillhörande konkreta och utförliga exempel.

Ta med delar från designdokument och implementation. För 2 och 3 behöver du naturligtvis använda någon del som ingår i en klasshierarki.