Forelesning 12: Genova

Innledning

Vi begynte kurset med å snakke om systemutvikling på overordnet prosess- og prosjekt-nivå, og fortsatte med kunde/leverandør/bruker-forhold. Deretter gikk vi inn i det mer tekniske ved systemets struktur, arkitektur og design (i UML). Nå har vi kommet fram til den siste delen av kurset som omhandler realiseringen av systemet til kjørbart produkt.

Generering av kode, database og brukergrensesnitt

Modellering (i for eksempel UML) hjelper systemutviklerne med å organisere og skape oversikt over systemet som skal lages. Siden UML er viden kjent og brukt i industrien, er slike modeller også en måte å dokumentere et system på en noenlunde standardisert måte, slik at (andre) systemutviklere kan videreføre arbeidet på systemet i fremtiden. Slik sett har UML-modeller en egennytte i seg selv, men UML-modellene kan også brukes videre til å generere det kjørbare systemet, og dette kan gjøres til dels automatisk. Dette fordrer kraftige verktøy som kan oversette UML-diagrammer til kjørbar kode med tilhørende databaser og grafiske brukergrensesnitt (skjermbilder).

Genova er et slikt verktøy. Genova baserer seg på såkalte domene-/informasjonsmodeller modellert i UML. En domene- eller informasjonsmodell er en modell som inneholder klassene og attributtene for en avgrenset enhet, for eksempel (system for) håndtering av utlån av bøker eller et bibliotek (som er en bedrift). For at Genova skal få vite hvordan et attributt skal fortone seg i en database og et brukergrensesnitt, er det lagt på en del egenskaper som kan settes på attributtet. Dette gjelder også noen andre UML-elementer, slik som klasse, attributt, assosiasjon og generalisering. I Rose settes egenskapene på i egne arkivkort på det gjeldende element, mens det for andre UML-verktøy er laget en UML-profil. En UML-profil inneholder egenskaper som følger stereotyper og hva slags UML-elementer stereotypene er gyldige for. Genova genererer grafiske brukergrensesnitt (skjermbilder), tjenester for å operere mot en database, grensesnittet til databasen og skjema for databasen basert på UML-modellene.

Det finnes en del velkjente måter å generere både programkoden og skjermbildene på, og Genovas tilnærming til dette, følger noen kjente arkitekturer, fra den overordnede n-lagsarkitekturen til mønsteret Model-View-Controller i brukergrensesnittet. Den vanligste realiseringen av n-lagsarkitektur, er trelagsarkitektur med entitetslag, kontrollag og kantlag (høres velkjent ut, ikke sant?), og Genova genererer i henhold til denne overordnede arkitekturen. I Genova-sammenheng kalles dette henholdsvis dataaksess, (dataorientert)

tjeneste og brukergrensesnitt. Som gjennomgått på tidligere forelesning(er), er den fysiske realiseringen uavhengig av de logiske lagene; man kan velge å realisere alle lagene på en datamaskin, inndelt etter lag eller også dele lagene på flere maskiner. Dette gir fleksibilitet for organisasjonen som skal drifte og forvalte løsningen etter produksjonsstart.

Mønsteret Model-View-Controller gir en vedlikeholdsvennlig inndeling i brukergrensesnittet (kantlaget), fordi de forskjellige delene har sitt spesifikke ansvar. Dette gjør for eksempel at man ikke får kode knyttet til når en knapp er trykkbar, blandet sammen med koden for hvordan elementene i brukergrensesnittet legges ut på skjermen.

Når det gjelder generering av databaser, så følger Genova prinsipper som ble gjennomgått på forelesningen om persistens og databaser. I denne forelesningen, vil vi se på hvordan man realiserer et databaseskjema og lager bindeverket mellom programmeringsspråket og databasen, slik at man får jobbet mot databasen fra systemet.

Systemer vil i det meste av sin levetid befinne seg i forvaltning og det er derfor viktig å tilrettelegge for dette. Det vil ofte være andre enn de som utviklet et system som vedlikeholder det. Modeller gir ofte en mer overordnet oversikt over et system enn tilsvarende kildekode og dette gjør det enklere for nye utviklere å sette seg inn i systemet. Videre er det slik at en modell er modellert i et modellspråk, som for eksempel UML, og selv om modellspråket bruker begreper fra programmeringsspråk, er modellen uavhengig av programmeringsspråket. Transformasjonen (genereringen) fra modell til kildekode vil derfor være like smertefri når programmeringsspråket er Java, som C#, Simula eller andre språk, så lenge malene som brukes til transformasjon er riktige. Dette betyr at arbeidet som er lagt ned i å modellere et domene kan høstes av i lang tid.

Dagens forelesning

Dagens forelesning vil ta for seg produksjon av kjørende system på bakgrunn av modeller. I og med at det skal gjøres en obligatorisk oppgave med Genova, vil majoriteten av tiden bli brukt på hvordan man bruker de forskjellige delene av Genova og hvilke resultater man får.

Relasjon til tema på neste forelesning

Dagens forelesning relaterer seg til tema neste gang ved at vi produserer resultater fra dagens forelesning, mens neste forelesning vil omhandle testing av de resultatene vi produserer; dette være seg enhetstesting, integrasjonstesting, systemtesting eller akseptansetesting