Göteborgs universitet

Institutionen för informatik

Systemutveckling
-en modell för verifiering och validering

Att utveckla system som motsvarar kundens krav lyckas inte alltid. Anledningen till detta kan vara att system inte verifieras och valideras under hela utvecklingen utan endast i slutet. Ett sätt att göra något åt detta problem skulle då kunna vara att skapa en självständig verifierings- och valideringsmodell för att kontinuerligt under utvecklingen försäkra att rätt system utvecklades och att det utvecklades på rätt sätt. Denna utmaning togs och genom litteraturstudier skapades en modell för denna uppgift. En jämförelse gjordes mellan två olika systemutvecklingsmetoder för att undersöka om det fanns några likheter eller skillnader mellan att använda dessa metoder med modellen som komplement. De slutsatser som framkom var att båda metoderna hade möjlighet att utveckla system som motsvarade kundens krav om de kompletterades med modellen. Även konstaterades det att modellen var ett bra komplement som uppmärksammade vikten med att utföra ett grundligt utvecklings-arbete.

Sara Eliason

Jenny Karnehed

Examensarbete I 10 p

ADB-programmet 80 p

Höstterminen 1999

Handledare:

Kjell Engberg

1. INLEDNING

1.1 Problemformulering

När man utvecklar system vill man att dessa ska vara av hög kvalitet. Det är tyvärr få människor som förknippar systemutveckling med kvalitet eftersom det idag förekommer system som inte uppnår de förväntningar som användarna har. En bidragande orsak till dålig kvalitet kan vara att systemutvecklare kastar sig över kodningen på en gång utan att tänka på vikten av förarbetet som analys och design.[1] Ofta skrivs ett program snabbt ihop och bristande planering måste kompenseras genom testning som utförs i slutet av utvecklingsarbetet.[2] Ett exempel på ett system som ej uppfyllde användarnas kravspecifikation, är när Stockholms stads parkeringsbolag Enator beställt ett ekonomisystem från ett dataföretag. Det visade sig först vid testningen av det levererade systemet att det ej stämde överens med kravspecifikationen.[3] Ett annat exempel är ett patienthanteringssystem på ett sjukhus som blev försenat p.g.a. att man först vid acceptanstesterna upptäckte att systemet inte hade den funktionalitet som det var avsett att ha. Funktionerna var helt enkelt inte tillräckligt utvecklade i enlighet med kravspecifikationen som skrivits.[4] Dessa två exempel visar att det utvecklas system som inte motsvarar kundens krav. Det kostar mycket att bygga om systemet samtidigt blir både systemutvecklare och kund irriterade över att systemet tar längre tid att bli färdigt än planerat.

Vissa av dagens systemutvecklare har uppfattningen att kvalitet hos en programvara uppnås mot slutet av utvecklingsarbetet genom testning. I denna fas görs försök att upptäcka så många defekter som möjligt på en nästan färdig programvara.[5] Visserligen förbättrars på detta sätt kvaliteten på den produkt som ska levereras till kunden, men test kan inte garantera felfria program.[6] Utvecklare kan testa defekter i åratal utan att vara säker på att ha hittat alla, eftersom testning endast hittar defekter utan att kunna bevisa dess frånvaro.[7] Kvalitet uppnås inte genom avslutande test utan kvalitetstänkande är något som måste genomsyra hela systemutvecklingen. Då man t.ex. ska landsätta en människa på månen och tryggt få honom tillbaka är man tvungen att tänka på kvaliteten i varje liten detalj.

Systemutvecklare måste bli mer ingenjörsmässiga för att göra system som erhåller hög kvalitet. En ingenjör inleder konstruktionen med omfattande planering och förarbete och sätter inte igång att bygga direkt.[8] Systemutvecklarnas testningar kan jämföras med en ingenjörs brobygge:

”...ingenjörer vet hur man bygger broar som håller. Under byggtiden inspekterar man bygget och ser till att specifikationerna följs. Man testar inte bron under byggtiden –man kör inte över den med en lastbilskaravan varje dag för att se om den håller.” (Lottson, 1995 nr. 41)

Ett ingenjörsmässigt jobb börjar med att arbetet planeras, man tar sig tid att tänka före, gör ett noggrant förarbete och prioriterar kvalitet framför allt annat.[9] När ett sådant

noggrant arbete gjorts ska allting fungera och man behöver ej testa att allt fungerar som det ska. Allt för omfattande testning under utvecklingsarbetet kan tolkas som att man inte litar på den utvecklingsmetod som använts för att utveckla systemet.[10] Man måste förstå innebörden av att det som är billigt att rätta till direkt, kan vara mycket dyrt att rätta till senare. Därför pratas det idag om att det är viktigt att lägga ner mer tid i systemutvecklingen på analys och design. För att utveckla ett system med hög kvalitet måste kvalitetstänkandet vara med under hela utvecklingsarbetet och inte endast fokuseras på testning.

Vissa företag förväntar sig högre kvalitet genom att utveckla system med objektorienterade metoder än att utveckla system med traditionella metoder.[11] För att uppnå kvalitet och därmed bättre system anser vi att metoden bör ha genomtänkta verifierings- och valideringsaktiviteter som startar vid början av utvecklingsarbetet och som används under hela utvecklingen.

1.2 Syfte

Syftet med denna uppsats är att förklara vikten med att använda verifierings- och valideringsaktiviteter under hela systemutvecklingen samt att göra en jämförelse för att undersöka om det finns några likheter och skillnader mellan att utveckla ett system med en objektorienterad metod än att uveckla ett system med vattenfallsmodellen, när de kompletteras med en självständig verifierings- och valideringsmodell.

1.2.1 Målgrupp

Det finns flera olika målgrupper som kan ha intresse av denna uppsats. I första hand riktar sig den sig till personer som ska börja arbeta med systemutveckling och som har föreställningen att de kan ”börja koda direkt”. Vi vill genom denna uppsats poängtera vikten av att planera och tänka igenom systemanalys och design innan kodningen tar fart, för att uppnå systemkvalitet.

Ytterligare en målgrupp är personer som arbetar med systemutveckling och som har den uppfattning att systemkvalitet uppnås genom att lägga stora resurser på verifierings- och valideringsaktiviteter mot slutet av systemutvecklingen genom testning. Vi har föreställningen att systemkvalitet uppnås genom att utföra verifierings- och valideringsaktiviteter kontinuerligt under systemutvecklingen, och inte endast under slutfasen genom testning.

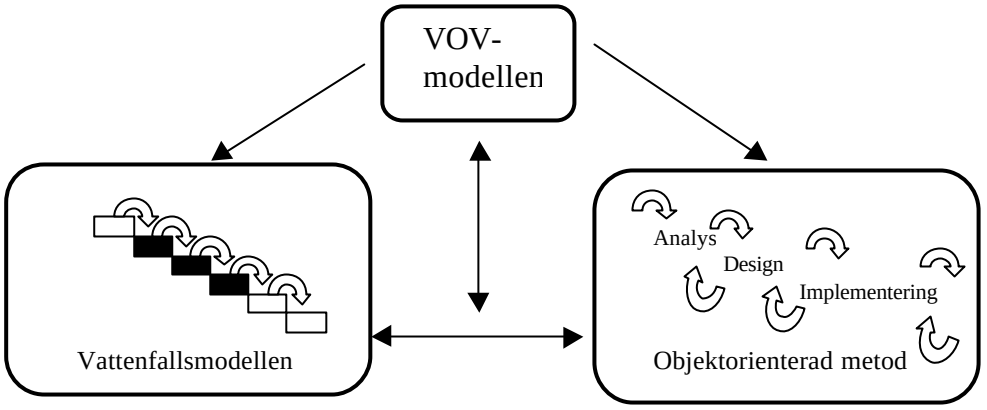
En annan målgrupp vi kan tänka oss är de som tror att systemkvalitet är något som uppnås genom att endast använda sig av en viss systemutvecklingsmetod. Vi gör en jämförelse där vi tar upp de skillnader och likheter som uppstår beroende på om systemet utvecklats med en objektorienterad metod eller mer traditionellt, enligt vattenfallsmodellen, när de kompletteras med en självständig verifierings- och valideringsmodell.

1.2.2 Vårt bidrag

Vårt bidrag är att i denna uppsats förflytta fokus från att systemkvalitet uppnås genom att verifiera och validera ett nästan färdigt system på slutet i utvecklingen, genom testning, till att fokusera på att systemkvalitet uppnås genom att kontinuerligt utföra väl genomtänkta verifierings- och valideringsaktiviteter under hela systemutvecklings-processen.

För att uppmärksamma att väl genomtänkta verifierings- och validerings-aktiviteter kan höja systemkvaliteten har vi skapat en modell som vi kallar VOV-modellen. Den första bokstaven ”v” har betydelsen verifiering och den andra validering, bokstaven ”o” har vi valt att placera mellan dessa med betydelsen och. På detta sätt får vi ett begrepp som både innefattar verifiering och validering utan att lämna någon av dessa två utanför. Vi anser att de båda tillsammans har viktiga betydelser för systemutveckling. Anledningen till att vi valt att utforma en modell är att vi vill beskriva det arbete som måste utföras, dvs verifiering och validering. Denna modell kan användas som ett komplement till den metod som valts att användas vid utvecklingen av ett system.

Genom att jämföra hur två olika typer av systemutvecklingsmetoder kan använda sig av VOV-modellen som komplement för att uppnå systemkvalitet vill vi förklara att det inte endast beror på metodvalet som systemkvalitet uppnås, utan att det även beror på hur verifierings- och valideringsaktiviteterna utförs.



Figur 1 En beskrivning av uppsatsens uppläggning.

1.3 Metod

1.3.1 Induktion, deduktion eller abduktion

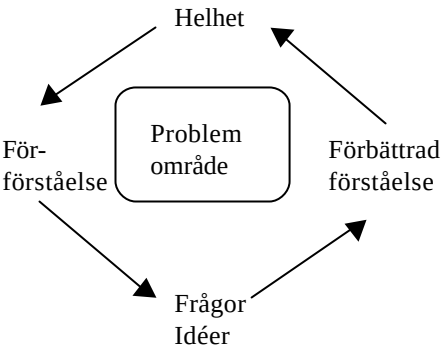
Förklaringsmodeller kan delas in i induktiva och deduktiva ansatser där induktiva ansatser har sin utgångspunkt i ett antal enskilda fall där gemensamma mönster söks i syfte att generalisera och där deduktiva ansatser utgår från en teori som ska förklara ett visst fall.[12]

Uppsatsen kan delas in i tre delar där den första delen ger en generell beskrivning av systemutveckling och kvalitet. I den andra delen skapas en VOV-modell innehållande validerings- och verifieringsaktiviteter. Vi har utformat modellen genom litteraturstudier där vi valt ut de aktiviteter som betraktats som de mest effektiva och utifrån dessa beskrivit dem ur vårt perspektiv och lagt till faktorer vi ansett vara relevanta. På så sätt har VOV- modellen skapats. Utformningen av modellen kan ses som en induktiv ansats. Den sista delen i studien innebär att dels beskriva hur två olika typer av systemutvecklingsmetoder kan kompletteras med VOV-modellen och dels att göra en jämförelse av de två beskrivningarna. Denna tredje delen i studien är en blandning av både teori och data. Denna kombination av induktion och deduktion kallas abduktion.[13] Vår uppsats kan sammanfattningsvis sägas ha en abduktiv ansats.

1.3.2 Vetenskapligt synsätt

De synsätt som dominerar forskningen är positivism och hermeneutik, vilka utgör två ytterligheter. Ett positivistiskt synsätt bygger på logiska resonemang och kvantitativa mätningar. Denna typ av forskning har varit central sedan 1700-talet men har under de senaste decennierna utsatts för kritik som bl.a. handlat om att det många gånger är fel eller omöjligt att öka kunskap genom empiriskt statistiska undersökningar. Istället vill man öka kunskap genom helhetsorientering och förklaringsmodeller där förståelse och tolkning av helheten är viktigare. Detta problematiserande synsätt kallas hermeneutik.[14]

Vi genomför vår uppsats med ett hermeneutiskt synsätt där vår egen kunskap och våra tankar används som en tillgång för att tolka den data som vi samlat in. Den hermeneutiska tolkningsmetoden brukar beskrivas som att forskaren utifrån en förförståelse för ett problemområde formulerar frågor och idéer till det problemområde som ska undersökas. De svar som forskaren får tolkas och leder till förbättrad förståelse, där nya frågor uppstår, ny dialog osv. Utifrån detta får man bättre förståelse för problemområdets helhet.[15]



Figur 2 Hermeneutisk tolkningsmetod.

1.3.3 Kvantitativa och kvalitativa studier

Studier brukar delas in i kvantitativa och kvalitativa beroende på hur information samlas in och bearbetas. Kvantitativa studier används vid mätningar av olika företeelser och när man söker svar på frågor som t.ex. Hur ofta förekommer något? Exempel på kvantitativa studier är när prover, enkäter och experiment används för att mäta olika företeelser. En kvalitativ studie ska ge en tydlig beskrivning av det som finns så att vi får en bättre förståelse av företeelser. Det kan innebära att samla in data från en eller flera fallstudier

och utifrån dessa generera en teori. Den kvalitativa studien innehåller inte mätningar med siffror eller tal.[16]

Vår uppsats kan karaktäriseras som en kvalitativ studie där vi samlar in data från ett avgränsat problemområde för att utforma VOV-modellen och beskriva hur denna kan användas för att komplettera den systemutvecklingsmetod man använder. Uppsatsen innehåller inga kvantitativa inslag, dvs. inga mätningar.

1.3.4 Saklighet, objektivitet och balans

En framställning av en studie ska sträva efter balans. Saklighet och objektivitet är två begrepp som ingår i balansbegreppet. Balans i en framställning innebär att man gett rätt utrymme till det som behandlas, att viktiga delar av studien ges större utrymme än till mindre relevanta delar.[17]

Den fakta som samlas in ska vara saklig, vilket innebär att fakta ska kontrolleras att vara riktig och korrekt. En huvudregel är att gå till primärkällan för att kontrollera en faktas saklighet. När fakta presenteras ska en strävan mot objektivitet föreligga. Forskaren bör vara medveten om sina egna fördomar och förutfattade meningar för att undvika att framställningen vinklas omedvetet. Forskaren ska även uppmärksamma om den fakta som insamlas är vinklad med hänsyn till källförfattarens intressen, något som kan vara svårt att upptäcka. För att uppnå objektivitet bör ståndpunkter från två olika håll återges.[18]

Inom systemutveckling finns det de som förespråkar objektorienterade metoder framför de mer traditionella metoderna, som vattenfallsmodellen.[19] I vår framställning strävar vi efter att så objektivt som möjligt framställa dessa båda sätt att utveckla system, utan att själva ta ställning för eller emot någon av metoderna.

1.3.5 Källkritik

För att uppnå saklighet och objektivitet i förhållande till sitt eget material används källkritik.[20] Det är viktigt att vara medveten om att värde, tillförlitlighet och aktualitet kan variera vid valet av skriftliga källor och därför bör källor värderas.[21] Ett sätt att göra ett urval av det material som samlats in kallas källkritik. För att känna till gränserna för det resultat som framkommit i studien ska det material som redovisas i studien värderas genom källkritik.

Ett material bedöms utifrån tre kriterier:[22]

- Samtidskrav – en källas värde minskar ju längre tid som gått sedan källan framställdes.
- Tendenskritik – ett övervägande för att värdera om källan framställts vinklad för att vinna gehör. Om så är fallet minskar källans tillförlitlighet.
- Beroendekritik – en källa bedöms om den påverkats av andra informationsgivare.

När vi söker material värderar vi det utifrån samtidskravet och har letat efter nyskrivna källor. Vi har bl.a. letat efter nyskrivna källor på Internet och när man använder sig av Internet som källa är det svårt att bedöma materialets kvalitet och trovärdighet. Den information som vi främst har använt oss av från nätet i uppsatsen kommer från Computer Sweden och vi anser att dessa artiklar har lika stor trovärdighet som de hade haft om de var publicerade i tidningen. Då vissa av författarna till de böcker vi läst förespråkar vissa systemutvecklingsmetoder framför andra, har vi granskat materialet genom tendenskritik och hämtat fakta från flera olika författare för att få en så nyanserad bild som möjligt inom ämnet. Vi har funnit litteratur som förespråkar vissa angreppssätt och har betraktat dessa källor utifrån beroendekritik och sett förbi vissa av dessa för att få ett så generell bild som möjligt.

Mycket av den litteratur vi använt har varit skriven på engelska som vi fritt översatt. Vissa av de engelska uttrycken har varit svåra att översätta till svenska och det finns vissa uttryck som vi valt att inte översätta utan behållit det engelska ordet, som t.ex. ”project scope”. De diagram vi ritat baseras på information som inhämtats från olika källor.

1.3.6 Primärdata och sekundärdata

Vid insamling av data brukar man skilja mellan primär- och sekundärdata. Primärdata erhålls utifrån t.ex. protokoll, offentlig statistik eller intervju medan sekundärdata är information som finns dokumenterad i t.ex. vetenskapliga uppsatser och doktors-avhandlingar. Båda sätten att samla in data kan användas var för sig eller i kombination.[23]

Vi har använt oss av sekundärdata i denna uppsats och hämtat information från litteraturstudier, tidningar och Internet. Vi har kartlagt befintlig litteratur inom problemområdet för att utifrån denna skapa en uppfattning inom ämnesområdet.

1.3.7 Reliabilitet och validitet

Trovärdigheten i en studies resultat kan bedömas utifrån de metoder och de källor man använt för att utföra studien. För att en studies forskningsresultat ska bedömas vara vetenskapligt måste de metoder och tekniker som används uppfylla kraven att vara valida och reliabla. Reliabiliteten i en viss metod är att bedöma användbarheten och tillförlitligheten av en viss metod och en metods validitet är ett mått metodens förmåga att mäta det studien avser att mäta så att inte fel saker mäts. Validiteten i en studie säkerställs genom att använda flera olika metoder på samma problem eller genom att hämta data från flera oberoende källor.[24]

Vi har höjt reliabiliteten i vår uppsats genom att hämta information från flera oberoende källor. Vi har valt att inte intervjua personer med anledning till den tidsbegränsning vi har haft och har istället lagt stor tyngdpunkt på litteraturstudier. Vi anser att litteraturstudierna har givit oss det stöd som behövs för att nå fram till vårt resultat.

1.4 Avgränsningar

Vi har valt att inte gå in på och beskriva vilka olika typer av systemutvecklingsverktyg som kan användas för att höja systemkvaliteten utan avgränsat oss till att beskriva hur verifiering och validering av delprodukter kan utformas under olika systemutvecklingsfaser för att erhålla systemkvalitet.

Systemutveckling kan delas in i olika faser beroende på vilken systemutvecklingsmetod som används.[25] I vår uppsats har vi avgränsat oss till att hantera verifierings- och valideringsaktiviteter som utförs under analys, design och implementering av system. Alltså berör vi ej förarbetet till analysfasen eller senare delar som underhåll av systemet som utvecklats. Vi betraktar systemutveckling som ett antal faser där analys, design och implementering ingår och där det är vanligt att faserna går in i varandra och har iterativa inslag.

Vilken typ av verifierings- och valideringsaktiviteter som utförs under utvecklingen av ett system skiljer sig beroende på vilken typ av system som utvecklas. Utveckling av ett renodlat tekniskt system som t.ex. programvara för flygplansradar har troligtvis en annan typ av verifierings- och valideringsaktiviteter än utvecklingen för ett administrativt system. Vi har begränsat vår studie till att endast innefatta administrativa system som t.ex. inköps-, bank- eller försäkringssystem.

Det finns flera olika typer av systemutvecklingsmetoder att välja mellan när ett system ska utvecklas. Valet av systemutvecklingsmetod beror bl.a. av vilken typ av system som ska utvecklas.[26] För att avgränsa vår uppsats har vi valt att beskriva hur två relativt olika metoder att utveckla administrativa system kan använda sig av VOV-modellen som komplement. Den första typen av system som vi valt beskriva är system som utvecklas enligt vattenfallsmodellen, som är ett traditionellt sätt att systemutveckla, och den andra typen är system som utvecklas med en objektorienterad metod. Vårt syfte med dessa beskrivningar är sedan att göra en jämförelse av hur man på två relativt olika sätt kan utveckla system och i båda fallen använda VOV-modellen som komplement för att uppnå systemkvalitet.

1.5 Rapportstruktur

Uppsatsen är indelad i tre delar. I den första delen vill vi ge läsaren en förståelse för systemutveckling, systemkvalitet och beskriver centrala begrepp. I den andra delen går vi igenom utformningen av den VOV-modell vi skapar och beskriver vilka verifierings- och valideringsaktiviteter den inbegriper samt i vilka faser dessa ska utföras. I den sista delen i uppsatsen går vi igenom och jämför hur två relativt olika systemutvecklingsmetoder kan använda VOV-modellen som komplement för att kvalitetssäkra de delprodukter som utvecklas. Vi gör sedan en jämförelse för att undersöka vilka likheter och skillnader användandet av VOV-modellen som komplement till de båda systemutvecklingsmetoderna medför.

1.5.1 Centrala begrepp i systemutveckling

Första delen av vår uppsats beskrivs de centrala begrepp inom systemutveckling som vi finner relevanta för att ge läsaren underlag att ta del av uppsatsen. Vi börjar med att beskriva generellt om systemutveckling och sedan beskriver vi vilka ekonomiska konsekvenser dålig kvalitet kan medföra. Vad det kostar att upptäcka fel och korrigera dessa sent i systemutvecklingen. Vidare beskriver vi ett antal definitioner av begreppet kvalitet. Ur dessa definitioner gör vi en sammanställning över vad vi anser vara kvalitet. Eftersom verifiering och validering är viktiga faktorer i uppsatsen, definieras även dessa, för att få en bra grund och förståelse för vad begreppen innebär.

1.5.2 VOV-modellens utformning

I den andra delen i uppsatsen beskrivs de verifierings- och valideringsaktiviteter som VOV-modellen inbegriper. Aktiviteterna är indelade i analys-, design- och implementering för att ge uppsatsen en logisk struktur. Vi beskriver samtidigt med verifierings- och valideringsaktiviteterna i varje systemutvecklingsfas vilka vanliga typer av fel som förknippas med de olika faserna. Detta för att uppmärksamma vilka typer av fel som verifierings- och valideringsaktiviteterna bör fokusera på att hitta och förebygga i respektive fas.

1.5.3 Användning av VOV-modellen

I den tredje delen i uppsatsen beskriver vi först hur det går till när ett system som utvecklas enligt vattenfallsmodellen kompletteras med VOV-modellen. Vi beskriver sedan på liknande sätt hur VOV-modellen används när den kompletterar utvecklingen av ett system som utvecklas med en objektorienterad metod. Ur dessa beskrivningar gör vi sedan en jämförelse för att undersöka vilka likheter och skillnader som uppstår när dessa metoder att systemutveckla kompletteras med VOV-modellen. De båda beskrivningarna är upplagda så att jämförelser mellan verifierings- och valideringsaktiviteterna lätt kan göras genom att underrubrikerna liknar varandra och ger beskrivningarna en logisk struktur. Beskrivningarna följer även VOV-modellens utformning så att läsaren vid behov lätt ska kunna gå tillbaka och titta i VOV-modellen.

2. CENTRALA BEGREPP I SYSTEMUTVECKLING

Detta kapitel syftar till att beskriva centrala begrepp i systemutveckling. Vi ger en beskrivning av hur systemutveckling kan delas in i olika faser och sätter ekonomiska aspekter i relation till systemutveckling och systemkvalitet. Vidare diskuteras begreppen kvalitet, verifiering och validering.

2.1 Systemutveckling

Systemutveckling beskriver det sätt man använder för att skapa ett informationssystem.[27] Systemutveckling kan definieras som *”Analys, utformning och förändring av verksamheter där informationssystem ingår eller förväntas ingå”* (Cronholm, 1998, s.50). Systemutveckling är en komplex företeelse och betraktas ofta som en process som är indelad i olika faser. Ett sätt att uppfatta denna uppdelning är att tala om olika abstraktionsnivåer. En vanlig fasindelning är följande:[28]

- Verksamhetsanalysfas
- Utformningsfas
- Realiseringsfas
- Implementeringsfas

I ovanstående indelning består systemutveckling av de fyra faserna verksamhetsanalys, utformning, realisering och implementering. Den produkt som produceras under analysfasen är en kravspecifikation. Under utformningsfasen produceras en ritning till ett realiserbart informationssystem som under realiseringsfasen förverkligas genom programmering av kod. Slutligen i implementeringsfasen förs systemet in i den miljö den ska arbeta.[29] I denna uppsats har vi valt att utgå från de tre första faserna i ovanstående fasindelning, samtidigt som vi har gett de olika faserna i systemutvecklingen andra namn.

Tabell 1VOV-modellens definitioner på systemutvecklingsfaser.

Vanligt förekommande faser		VOV-modellens definitioner
Verksamhetsanalysfasen	→	Analysfasen
Utformningsfasen	→	Designfasen
Realiseringsfasen	→	Implementeringsfasen
Implementeringsfasen	→	Underhållsfasen

De faser som vi behandlar i uppsatsen är analys-, design- och implementeringsfasen. Vi är medvetna om att underhållsfasen är en viktig del i systemutvecklingen men vi har ändå valt att lägga tyngdpunkten på de tidigare faserna.

Den fundamentala aktiviteten i systemutveckling är programmering vilket ingår i implementeringsfasen. Programmering går ut på att skriva instruktioner som kan läsas och utföras maskinellt på en teknisk plattform. Men innan programmering börjar måste systemet förstås och beskrivas. En viktig del i systemutvecklingen är att beskriva verksamheter och deras informationssystem, samt att göra analyser med utgångspunkt från dessa beskrivningar.[30] Syftet med dessa delar av systemutvecklingen, som vi i vår uppsats kallar analysfas och designfas, är att skapa överblick över kraven på systemet och att fastlägga grundvalen för realiseringen av systemet. Analysfasen kan sägas vara den viktigaste fasen inom systemutveckling och måste genomföras på ett tillfredsställande sätt. Det är omöjligt att kompensera ett dåligt analysarbete med ett bra jobb i de följande faserna.[31] Detta beror på att senare utvecklingsarbete bygger på vad som producerats under analysfasen. Syftet med analysfasen är att förstå vad kunden, användarna, verksamheten behöver och syftet för designen är sedan att utifrån dessa behov göra en ritning över hur systemet ska fungera. Implementeringsfasens syfte är slutligen att realisera designen av systemet på en teknisk plattform.[32]

Systemutveckling är en process som involverar människor med olika bakgrund och kompetens. Utvecklingsprocessen karaktäriseras ofta som dynamisk, situationsanpassad och iterativ. Systemutvecklarna väljer de metodkomponenter som är relevanta för utredningssituationen dvs arbetssätt, notation och begrepp.[33] Olika organisationer använder olika systemutvecklingsmetoder för att utveckla system och vissa metoder passar bättre än andra för vissa typer av system. Om fel metod används, minskas troligtvis kvaliteten eller användbarheten av det system som ska utvecklas.[34] Men kvaliteten avgörs inte bara genom själva användningen av en metod utan det avgörande ligger i hur tekniken används.[35] Det finns ett antal olika generella metoder för systemutveckling nedan beskriver vi två vanliga sätt:

- *Vattenfallsinställning*: I denna inställning representeras systemutvecklings-aktiviteterna som separata processfaser som följer efter varandra, exempelvis kravspecifikation, design, implementation, testning osv. Först efter att varje steg är klart avskrivs det och utvecklingen går vidare till nästa fas.[36]

- *Objektorienterad inställning*: Vid användandet av detta angreppssätt så diskuteras vilka förhållanden i och utanför verksamheten det är önskvärt att ha information om. Dessa förhållanden kallas objekt och det kan ta emot ett meddelande, bearbeta det efter en viss mall och sända iväg det till andra objekt. I varje fas tänker man objektorienterat och övergången mellan faserna är mjuk.[37]

Dessa två metoder som vi beskrivit ovan har vi valt att studera i denna uppsats. Vattenfallsmodellen är en av de enklaste metoderna att använda vid systemutveckling och är den metod som varit mest vanligt förekommande samtidigt som objektorienterade metoder fått mycket uppmärksamhet den senaste tiden. Förespråkare till

objektorienterade metoder hävdar att objektorienterade system är lättare att bygga och underhålla samt att man lättare kan röra sig mellan de olika systemutvecklingsfaserna. Vattenfallsmodellen passar dock bra att använda när kunden från början vet vilken typ av system de vill ha och när systemutvecklarna vet vad som går att realisera.[38]

Under systemutvecklingen ska systemutvecklare producera ett system som ska levereras till kunden tillsammans med dokumentation som beskriver hur man ska installera och använda systemet.[39] Men systemutveckling handlar inte bara om att producera system och dokumentation utan att producera dem på ett kostnadseffektivt sätt. Utmaningen för systemutvecklare är att producera hög-kvalitativ mjukvara med ett begränsat antal resurser genom att använda ett förutbestämt schema.[40] Får man obegränsat med resurser, skulle majoriteten av problemen med felaktiga system troligtvis klaras ut. Men kvalitet och systemutveckling har inte alltid förknippats med varandra, systemutveckling förknippas av många som något som drar ut på tiden, drar över budget och ofta något som inte blir vad kunden önskar.[41] Enligt åtskilliga undersökningar är de oftast förekommande problemen med informationssystem och deras utveckling följande fyra punkter:[42]

- Missnöjda användare
- Dyr och tidsödande utveckling
- Dyrt och tidsödande underhåll
- Otillförlitlighet

Dessa problem kan åtgärdas om det vid utvecklingen av informationssystem tas större hänsyn till både kvaliteten vid produktionen och kvaliteten vid användandet av informationssystem.[43]

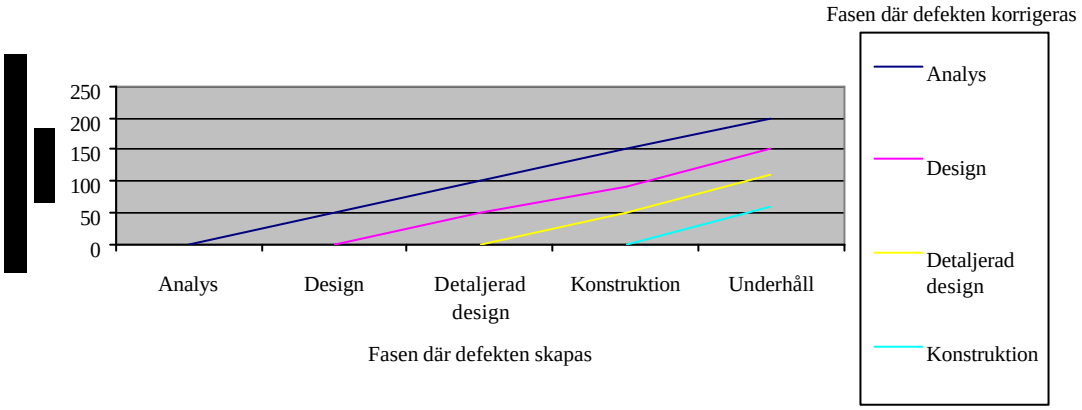
Eftersom orsaken till att flera av de systemutvecklingsprojekt som misslyckats har sin grund i att systemet inte motsvarar kundens krav och är av dålig kvalitet, behövs en bättre förståelse av själva systemutvecklingsprocessen. Varje företag måste ställa sig ett antal frågor såsom hur lämpliga deras systemutvecklingsmodeller, metoder, tekniker och verktyg är för att bemöta kvalitetsvärden. Men det är svårt att bedöma om den valda metoden för att utveckla ett system i sig säkrar systemkvaliteten. Vi anser att om det färdiga systemet ska vara av kvalitet så måste man kontinuerligt under system-utvecklingen utföra ett antal kvalitetssäkrande aktiviteter för att kontrollera de delprodukter som skapas under de olika utvecklingsfaserna. Genom verifiering och validering av delprodukter som produceras kommer kvalitet att uppnås. För att ge dessa kvalitetssäkrande aktiviteter mer uppmärksamhet anser vi att verifierings- och valideringsaktiviteter bör utformas till en självständig aktivitet som kompletterar den systemutvecklingsmetod som används vi systemutvecklingen.

2.1.1 Ekonomiskt perspektiv

Året 1995 gjorde analysföretaget Standish Group en granskning av 8000 projekt och kom fram till att endast 19 % av projekten blivit klara i tid med den budget man utgått från. De resterande projekten lades ned eller sprängde både tids- och resursramar. Förklaringen till

detta var enligt Standish Group en bristande hantering av användarnas krav. Under projektens gång ändrades kraven och trots att kraven dokumenterats kände inte användarna igen sig i sina gamla krav.[44] Med utgångspunkt från denna granskning som Standish Group utförde kan vi konstatera att det inte är många projekt som klarar av att utveckla system, med den budget, de resurser och den tid de har tillförfogande. Det gäller att ha en bra kontakt med användarna och under arbetets gång försäkra sig om att utvecklingen följer kravspecifikationen och att den är korrekt. Utvecklingen är som en vansklig balansgång eftersom allting ska vara färdigt fort samtidigt som systemet ska ha tillräckligt hög kvalitet för att sälja.[45] Förbättring av kvaliteten och minskning av kostnaderna på system måste nästan vara ett av målen inom systemutveckling. Får man obegränsade resurser kommer majoriteten av mjukvaruproblem troligtvis lösas.[46]

Kostnaden av att korrigera fel i olika faser är varierande och beror på när felet upptäcks och korrigeras.[47] Undersökningar har visat att fel som införs i de tidiga faserna i utvecklingen som t.ex. analys- och designfasen kostar 50 till 200 gånger så mycket mer att korrigera senare i utvecklingen, än om felet ändrats strax efter att det blivit inlagt.[48] Figur 3 beskriver hur kostnader ökar i relation till hur länge ett fel förblir oupptäckt.



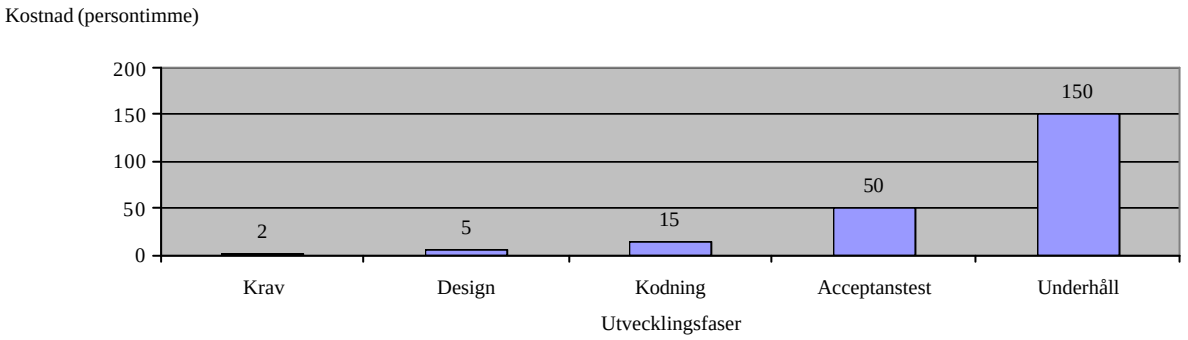
Figur 3 Ökning av defektkostnad i tid mellan skapande/korrigering av defekter (Källa: McConnell, (1998), s 29).

Figur 3 visar att om felen upptäcks i de senare faserna så ökar kostnaderna ordentligt för att korrigera dessa. Ur diagrammet kan även utläsas att om ett fel introduceras under analysfasen och det korrigeras först vid konstruktionsfasen, så kan det kosta upp till 150 gånger mer än om man hade korrigerat felet direkt under analysfasen. En sådan kostnad borde vara möjlig att minska, genom att försöka hitta och korrigera fel tidigare i utvecklingen. Helst redan under analysfasen. En mening i kravspecifikationen kan lätt bli flera diagram under designfasen. Senare i projektet, görs dessa diagram till flera hundra rader av kod, flera testfall, flera sidor av slutanvändarmanualer osv. Om system-utvecklaren har möjlighet att ändra misstag som görs vid analysfasen, när det enda jobbet som gjorts är att skapa en mening i ett krav, är det bättre att korrigera meningen än att göra det senare i utvecklingen.[49] Om det tidigare utvecklingsarbetet utförs på ett bra sätt blir det slutliga resultatet lyckat, men utförs det dåligt påverkar det hela utvecklingen.

”De största felen ligger alltid i kravspecifikationen, inte i koden.” (Lidfeldt, 1998, nr. 7) Med hänsyn till detta citat måste det vara universalt att hitta felen som inträffar i just den fasen där kravspecifikationen produceras och att inte vänta till testning för att hitta felen. Tyvärr görs detta inte i praktiken utan där koncentreras upptäckandet av fel vid testning. Upptäckandet av fel och korrigering av dessa borde göras under hela processen och att hitta fel direkt efter att de introducerats. För att minska det totala antalet av defekter som existerar i systemet vid leverans och att minska kostnaderna av att ta bort defekter, är en självklar inställning för att förebygga att fel introduceras.[50]

Kvaliteten på kravspecifikationen påverkar kvaliteten på det slutliga systemet och kostnaderna för systemutvecklingen. Den kritiska roll som kravspecifikationen spelar under ett systemutvecklingsprojekt borde vara de tidigare faktorerna som diskuterats. Man kan få för sig att varje systemutvecklingsprojekt har en hög kvalitativ kravspecifikation som startpunkt men tyvärr är det inte så i verkligheten. Detta kan bero på den bristande förståelsen av rollen och viktigheten med kravspecifikationen. Viljan att skynda på systemutvecklingen och skära i kostnaderna genom att ta bort icke väsentliga aktiviteter leder till att många systemutvecklingsprojekt börjar med en lågkvalitativ kravspecifikation som inte är fullständig utan full av tvetydigheter.[51] Att utveckla en kravspecifikation med låg kvalitet och sedan bygga på detta rangliga bygge, ökar även underhållskostnaderna.[52]

För att se hur viktigt det är att korrigera kravfel tidigt i utvecklingen, visas figur 4. Den visar hur många persontimmar som kan tjänas in genom att korrigera felen så tidigt som möjligt i utvecklingen:



Figur 4 Kostnad i persontimmar att korrigera kravfel (Källa: Jalote, (1997), s. 77).

Genom att minska felen i kravspecifikationen kan det medföra en stor kostnadsminskning för utvecklingen. Genom att investera i ytterligare 100 persontimmar i kravfasen, kan ett medeltal på omkring 50 nya fel upptäckas och tas bort. Om dessa fel ej upptäcks i analysfasen kommer de att upptäckas i en senare fas.[53]

Vi vet att krav ändras under systemutvecklingens gång. Om kravspecifikationen inte analyseras och inte tillräckligt med arbete lagts ner på att validera kraven leder detta till många förändringar av kraven i ett senare skede. Det är uppskattat att mellan 20 och 40% av den totala utvecklingsinsatsen i ett systemutvecklingsprojekt används till omarbetning,

där mycket av omarbetningen görs för att göra ändringar i kravspecifikationen.[54] Ej planerad omarbetning beräknas ta omkring 80% av tiden på hela projektet. Genom att planera omarbetning så kan kvaliteten och produktens produktivitet förbättras.[55] Att modifiera systemet tar vanligtvis upp till 60% av den totala kostnaden för systemet.[56] En kravspecifikation med hög kvalitet minskar kostnaderna både i arbetstimmar och kronor.

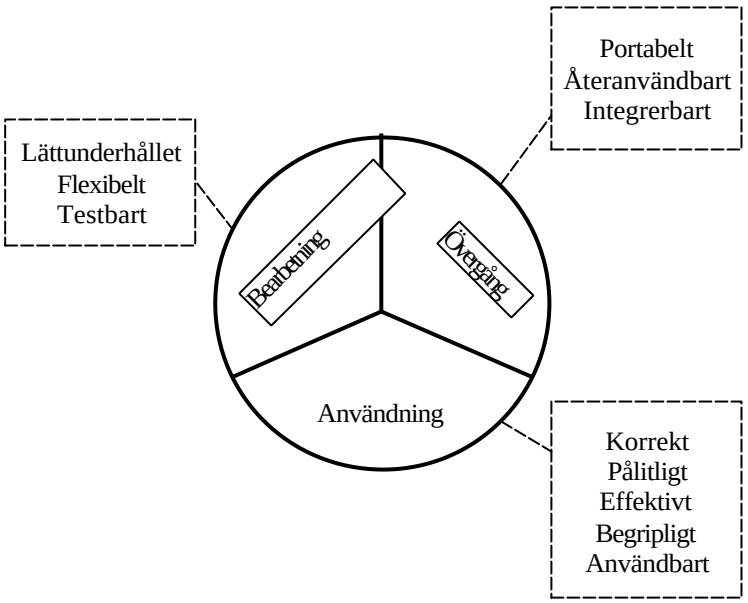
Kostnaden för dålig kvalitet uppgår i många företag till ansenliga belopp. I denna kostnad ingår oredovisade kostnader som är en följd av att chefer, inköpare, konstruktörer, försäljare m.fl. får ägna en hel del tid åt verksamhet som egentligen har sin grund i brister i kvaliteten som exempelvis omplaneringar, kunddiskussioner, konstruktionsändringar och sammanträden. Dessa oredovisade och därmed dolda kostnader är betydande.[57] Kvalitetskostnaderna inbegriper både kostnader för att uppnå en viss kvalitetsnivå och kostnader som följer av dålig kvalitet.[58] Genom att använda VOV-modellen minskas de kostnader som uppstår till följd av dålig kvalitet på de specifikationer och dokument som produceras, då användningen av VOV-modellen ska tillförsäkra god kvalitet på de dokument som produceras.

Sammanfattningsvis kan vi komma fram till att det är viktigt att hitta och korrigera fel så snart som möjligt under hela systemutvecklingsprocessen. Kravspecifikationen måste vara välgjord för att man ska nå lyckade resultat. Genom att göra detta minskas kostnaderna och man producerar ett system med god kvalitet.

2.1.2 Kvalitetsbegreppet

Kvalitet är ett begrepp som har varierande betydelse för människor. Vad menar vi när vi säger att ett system är av god eller hög kvalitet? Vad är kvalitet och hur produceras det? Kvalitetsbegreppet kan t.ex. referera till att det inte förekommer några avbrott i systemet eller till kommunikationen mellan mjukvaran och användarnas förväntningar.[59] Flera försök har gjorts för att beskriva de olika aspekterna av systemkvalitet. Kvalitet kan ses ur ett perspektiv där det består av de tre dimensionerna bearbetning, övergång och användning. Samtidigt som kvalitet ofta analyseras till ett flertal olika kriterier, som t.ex. korrekt, pålitligt, effektivt, begripligt, användbart, lättunderhållet, testbart, flexibelt, portabelt, återanvändbart och integrerbart. De första fem kriterierna relaterar till användningen av mjukvaran och de följande tre syftar till modifiering och bearbetning av mjukvaran med tanke till nya krav. De sista tre hanterar överföringen av mjukvaran för att fungera i nya miljöer.[60] Figur 5 visar de tre dimensionerna och de olika kriterier kvalitetsbegreppet kan inneha.

Kriteriet ”korrekt” är ett mått på om de uppställda kraven är uppfyllda medan exempelvis ”lättunderhållet” är ett mått på kostnaden för att hitta och rätta fel i systemet när det används. Ett annat kriterium som vi kan se i figur 5 är ”portabelt” vilket är ett mått på kostnaden för att flytta systemet till andra tekniska plattformar.[61] Problemet här är att om en av dessa kriterier ska fungera så kan det innebära att en annan kvalitetsfaktor inte kan uppfyllas.



Figur 5 De tre dimensionerna av kvalitetskriterierna (Källa: Jalote, (1997), s.11)

Ett exempel är när man strävar efter att uppnå effektivitet, vilket är ett mått på utnyttjandet av resurserna i den tekniska plattformen, så kan det bli svårare att erhålla bra underhåll och flexibilitet där flexibilitet är ett mått på kostnaden för att ändra i systemet när det är igång.[62] Vid ett sådant tillfälle får man helt enkelt bestämma sig för om det är effektivitet, underhåll eller flexibilitet som är viktigast i denna situationen och vilket av dessa begrepp som anses vara kvalitativt. Systemutvecklaren kan vara nöjd med kvaliteten på systemet medan kunden inte är nöjd med dess kvalitet vid användandet. Systemutvecklaren kan gömma sig bakom specifikationen och påstå att kunden inte har några grunder för att klaga eftersom de får det de efterfrågade, medan kunden kan klaga på att specifikationen inte hade tillräckligt bra kvalitet.[63]

Kvalitetens betydelse i dagens konkurrensinriktade marknad är självklar och behöver därför inte motiveras, men vad vi förstår så är det inte självklart vad man anser med begreppet kvalitet och därför måste det diskuteras. Systemkvalitet är ett begrepp med ett flertal innebörder som inte är lätt att definiera och därför uttrycks kvalitet på många olika sätt.[64] Vi presenterar nedan ett urval av dessa för att visa begreppets bredd:

- *”Kvalitet är ett mångdimensionellt begrepp som måste betraktas ur en rad perspektiv för att få en heltäckande bild. Ordet kvalitet bör användas med viss försiktighet eftersom risken för missförstånd är påtaglig. Det är svårt, och inte alltid ens önskvärt, att hitta en universell kvalitetsdefinition. Kvalitetsutveckling bygger på att kvalitet i första hand definieras utifrån kunden.”* (Blomqvist & Haeger, 1996, s. 26)

- *”Kvalitet är överensstämmelsen mellan den produkt man tar fram och en preciserad utgångspunkt. Utgångspunkten kan vara de förväntningar som finns på produkten, de behov produkten är tänkt att uppfylla eller en kravspecifikation.*” (Andersen, 1994, s.469)

- *”The quality of one and the same system is normally judged differently by the actors involved in its production and use.”* (Mathiassen, Munk-Madsen, Nielsen, & Stage, 1998, s. 138)

- *”Kvalitet är ett subjektivt begrepp som därför inte går att definiera entydigt, det betyder olika saker för olika människor. Kvalitet är i de flesta organisationer ingen garanti för framgång, däremot blir det alltmer en förutsättning för att hänga med i utvecklingen”* (Blomqvist & Haeger, 1996, s. 47)

- En definition som används av den svenska terminologistandarden ISO 9000, lyder: ”*…alla sammantagna egenskaper hos ett objekt eller en företeelse som ger dess förmåga att tillfredsställa uttalade eller underförstådda behov*.” (Jönsson, 1995, s. 6, Sandholm, 1992, s.12)

- Institutionen för kvalitetsutveckling (SIQ) som bl a administrerar *”Utmärkelsen Svensk Kvalitet”* väljer att låta bli att definiera begreppet: *”Kvalitet betyder i grunden hur något är – varans, tjänstens eller processens egenskaper. Varje organisation och medarbetare måste utifrån sina kunder bestämma vad man skall lägga i begreppet och formulera en definition som passar den egna verksamheten*” (Blomqvist & Haeger, 1996, s. 29)

- Ytterligare ett sätt att definiera kvalitet görs i följande citat: *”…företagets totala förmåga att i samverkan med andra företag tillfredsställa kundernas behov och att överträffa deras förväntningar”* (Fransson, & Quist, 1995, s. 8)

- *”Ordet kvalitet har sitt ursprung i latinets qualitas (av vad). Det har grundbetydelsen egenskap eller karaktärsdrag.”* (Jönsson, 1995, s. 6)

Utifrån de ovanstående beskrivna definitionerna på kvalitet har vi gjort en sammanställning av det vi utläser av begreppet kvalitet:

- Kvalitet kan vara något *subjektivt*. Det finns likheter mellan definitionerna och en likhet är att kvalitet tolkas olika av olika personer. Kvalitet kan betyda en sak för exempelvis en kund medan det betyder något annat för en systemutvecklare.
- Kvalitet kan vara att *möta en standard*.
- Tillgodose kundens uttalade eller outtalade *behov*.
- Kvalitet kan vara något som uppstår i *relation mellan kund och produkt*.
- Kvalitet avgörs utifrån användarnas *förväntningar*.
- Kvalitet är *egenskaper* hos produkten.
- Man måste betrakta kvalitet utifrån olika perspektiv.

Vi anser att det är bra att systemutvecklare tillsammans med kunden bestämmer och kommer överens om vad kvalitet innebär och lämnar därför begreppet öppet. Den bästa lösningen är att kombinera kundens kvalitetsbegrepp med systemutvecklarens. Vi rekommenderar att systemutvecklaren för en diskussion med kunden om vad kvalitet betyder för dem och sedan definierar begreppet gemensamt. Det är viktigt att systemutvecklaren förstår vad kunden menar med kvalitet, om kvalitets begreppet innebär användbarhet, flexibilitet eller om det har någon annan betydelse?

Arbetet med kvalitet bör utgå från kundernas behov, önskemål och krav. För att utveckla ett system av kvalitet måste systemutvecklare och kund börja med att utveckla en fullständig kravspecifikation. Det är sedan viktigt att systemutvecklarna under hela utvecklingsarbetet följer kravspecifikationen och behåller kontakten med kunden och för en diskussion angående kraven. De produkter som utvecklas under systemutvecklinges olika faser ska motsvara kravspecifikationen för att det slutliga systemet ska kunna uppfylla kundens förväntningar och krav. Det är därför viktigt att kontinuerligt under systemutvecklingen verifiera och validera de delprodukter som produceras.

2.2.1 Verifierings- och valideringsbegreppen

Verifiering och validering är centrala begrepp i denna uppsats och vi finner det därför viktigt att diskutera och beskriva begreppen. Det finns olika definitioner på vad verifiering och validering innebär och för att ge ett exempel beskriver vi två olika definitioner av begreppen och redogör sedan för vad definitionerna innebär i denna uppsats.

Lite slarvigt beskrivs ibland verifierings- och valideringsaktiviteter som om de vore samma sak. Men i en mer noggrann beskrivning skiljs verifierings- och valideringsaktiviteterna åt. Nedan följer ett citat på dessa två aktiviteter (Boehm, 1981, s. 728) :

- ”Verification: *To establish the truth of correspondence between a software product and its specification. (from the Latin veritas, ”truth”).”*

  ”Verification:”Are we building the product right?””

- *”Validation: To establish the fitness or worth of a software product for its operational mission (from the Latin valere, ”to be worth”).”*

  ”Validation:”Are we building the right product?””

Utifrån denna definition kan vi utläsa att verifiering innebär att kontrollera att systemet tillmötesgår specifikationen och att produkten byggs på rätt sätt. Medan validering innebär en kontroll på att systemet som implementerats möter kundens behov och att rätt produkt byggts.

En andra definition vi valt att presentera följer enligt nedanstående citat: (Arthur, Gröner, Hayhurst & Holloway, 1999, s. 79)

- *”Verification refers to the process of examining each development phase to ensure that the output of a particular phase satisfies all the pertinent requirements of the previous phase, is internally acceptable, and can support the development effort in the next fase.”*

- *”Validation, on the other hand, is an activity primary concerned with software testing. During validation you execute the system and compare the test result to the requirements.”*

Vad vi kan utläsa utifrån denna beskrivning av begreppen är att verifiering undersöker varje utvecklingsfas för att försäkra att dokumenten som kommer från en speciell fas motsvarar alla relevanta kraven från den tidigare fas och att de stödjer utvecklingsarbetet i kommande fas. Validering verkar ses som en aktivitet som används under tesningsfasen. Att man kör systemet och jämför testresultatet med kraven.

Dessa två definitioner som vi har beskrivit skiljer sig lite ifrån varandra, så därför anser vi att det bästa är att bena ut begreppen och ge en beskrivning för vad verifiering och validering innebär i denna uppsats.

Den första definitionen av verifiering beskriver endast att systemet ska kontrolleras med avseende på att den följer sin specifikation, dvs att produkten byggs på rätt sätt. Den andra definitionen ger en mer detaljerad beskrivning där verifiering innebär att alla specifikationer som utarbetats i en fas är konsistenta med specifikationerna som producerats i den föregående fasen. Utifrån detta kan vi komma fram till att genom verifiering av den dokumentation som produceras under de olika utvecklingsfaserna ska leda till att dokumentationen blir konsistent och tillförlitlig under hela utvecklingen. Detta för att förhindra att vidare arbete grundar sig på felaktiga uppgifter.

Skillnaden mellan de två definitionerna av validering är att den senare definitionen relaterar begreppet till testning för att se att systemet motsvara kraven medan den första definitionen inte talar om hur validering ska utföras. Denna första definition uttalar att produkten ska motsvara målen med systemet. Om målen finns beskrivna i kravspecifikationen får dessa två beskrivningar samma betydelse, alltså att validering innebär att kontrollera att systemet uppfyller det behov systemet var avsett att fylla.

Utifrån de ovanstående beskrivna definitionerna på verifiering och validering väljer vi att definiera de båda begreppen som två olika aktiviteter. Verifiering är den aktivitet som avgör om de produkter som utvecklas tillmötesgår de specifikationer som lagts i föregående fas. Vi *verifierar* om de delprodukter som utvecklas motsvarar de riktlinjer som legat som underlag till delproduktens utformning och svarar på frågan *”Bygger vi systemet rätt?*”. Validering är den aktivitet som avgör om den delprodukt som utvecklas är det som kunden önskar. Denna kontroll görs i en öppen kommunikation mellan

systeutvecklare och kund. Vi *validerar* om delprodukten motsvarar kundens krav och låter kunden svara på frågan ”*Bygger vi rätt system?*”.

Systemutvecklare och kund ska komma överens om var och när kontroller utförs och hur ofta detta är nödvändig. Verifierings- och valideringsaktiviteter bör ske redan i början när ett system utvecklas. Detta för att säkerställa att avvikelser på en delprodukt upptäcks på ett tidigt stadium. [65]

3. VOV-MODELLENS UTFORMNING

I detta kapitel beskrivs VOV-modellens grundtankar och VOV-modellens indelning i olika faser. Vidare beskrivs de verifierings- och valideringsaktiviteter modellen inbegriper och hur de ska utföras under respektive fas. Vanliga fel som förekommer under de olika faserna uppmärksammas och aktiviteterna fokuseras på att upptäcka och förebygga dessa fel. Milstolpar definierar vi som de delprodukter som produceras under systemutvecklingens olika faser. I VOV-modellen läggs stor vikt vid att kontinuerligt verifiera och validera dessa milstolpar som produceras under hela systemutvecklingen och har som mål att förflytta fokus från att endast utföra detta mot slutet av utvecklingen vid testning.

3.1 Vikten av verifierings- och valideringsaktiviteter

VOV-modellen ska användas som en självständig aktivitet parallellt med den systemutvecklingsmetod som valts att användas. På så vis ges verifiering och validering större uppmärksamhet och fungerar som ett komplement till den utvecklingsmetod man valt att arbeta med. Genom att använda VOV-modellen ges verifierings- och valideringsaktiviteter under analys- och designfaserna lika stor uppmärksamhet som vid implementeringsfasen. Orsaken till denna uppmärksamhet om att förflytta fokus handlar om att det är allmänt känt att upptäckandet av fel sent i utvecklingen ger större kostnader än att hitta fel tidigare i utvecklingen. (Se Ekonomiskt perspektiv kap. 2.1.1)

I de utvecklingsmodeller som finns existerar verifierings- och valideringaktiviteter oftast kontinuerligt under hela utvecklingen, men p.g.a. deadlines och strama budgetar inses inte vikten av att använda verifiering och validering utan utvecklingen går fort vidare, utan att ha kvalitetssäkrat milstolpen.[66] Det kortsiktiga tänkandet resulterar i att det system som utvecklats innehåller fel som upptäcks i ett sent stadium, som i värsta fall kan innebära att större delen av utvecklingsarbetet måste omarbetas, vilket blir kostsamt (Se ekonomiskt perspektiv kap. 2.1.1). VOV-modellen genomsyras av ett långsiktigt tänkande vilket innebär att kostnaderna vid de tidigare faserna i utvecklingsarbetet blir större, men att produkten som utvecklas ges god kvalitet och kostnaderna minskas på lång sikt.

VOV-modellen är strukturerad i de tre systemutvecklingsfaserna analys-, design- och implementering. Under dessa tre faser produceras olika milstolpar som ska verifieras och valideras enligt VOV-modellen för att kvalitetssäkra systemet. I tabell 2 deklareras hur VOV-modellen definierar de milstolpar som produceras under systemutvecklingen.

Tabell 2 VOV-modellens definition på de milstolpar som produceras

Analysfas	Designfas	Implementeringsfas
Kravspecifikation	Systemdesign	Programkod
	Detaljerad design	

Tanken med VOV-modellen är att denna ska komplettera den utvecklingsmodell som används med åtgärder som garanterar kvaliteten. Åtgärder som är centrala i VOV-modellen för att uppnå kvalitet är manuella genomgångar, utprovning av prototyper samt automatiserade tester. I VOV-modellen innebär manuella genomgångar att dokument som produceras under systemutvecklingen manuellt granskas och automatiserade tester innebär en utprovning av programvaran.

En övergripande beskrivning av hur VOV-modellen är konstruerad ges i tabell 3 där de streckade områdena innebär att det inte förekommer några åtgärder. Ett exempel på vad som kan utläsas är att utprovning av prototyper endast görs vid analysfasen och inte vid designen eller implementeringen.

Tabell 3 En övergripande beskrivning av VOV-modellen.

	Utvecklingsfas		
VOV-aktivitet	Analys	Design	Implementering
Manuella genomgångar	Kravspecifikation	Systemdesign Detaljerad design	Programkod
Utprovning	Prototyp	------	------
Automatiserade tester	------	------	Programkod

I VOV-modellen utförs manuella genomgånger i följande faser:

- Analysfas
- Designfas
- Implementeringsfas

VOV-modellen har flera typer av manuella genomgångar, de vanligast förekommande kallas inspektioner och genomgångar. Inspektionernas utformning grundar sig på de inspektioner som Fagan utvecklade på 70-talet.[67] Det finns många olika varianter på att utföra inspektioner beroende på vilken typ av system som utvecklas och vilken organisation man jobbar inom. Dock har inspektioner gemensamma drag och utifrån dessa riktlinjer har vi utformat VOV-modellens inspektioner. Förutom dessa riktlinjer har vi specificerat uppgifter som vi anser vara relevanta för att göra en bra inspektion, på ett mer detaljerat sätt, om man jämför med den litteratur som finns om inspektioner. Inspektioner skiljer sig från genomgångar på det sättet att inspektioner måste schemaläggas och följas upp medan genomgångar är mindre formella och kräver inte lika mycket planering. Tanken från början var att endast använda inspektioner men vi har valt

att även använda genomgångar då dessa kan ske mer spontant och kräver då inte lika mycket resurser.[68]

Utprovning av prototyper utförs under VOV-modellens analysfas. Prototyper utvecklas i nära kontakt med användare och därför anser vi att det är bra att arbeta med prototyper i de situationer då användarna inte riktigt vet vilka krav de kan ställa på det system som de behöver. Genom prototypen kan systemutvecklare och användare i samarbete komma fram till en kravspecifikation som går att realisera.

Automatiserade tester utförs vid verifiering och validering under VOV-modellens implementeringsfas. Systemet testas under denna fas med olika indata för att kontrollera om systemet fungerar enligt intentionen. Automatiserade tester är den typ av verifierings- och valideringsaktiviteter som ges mest uppmärksamhet i den litteratur vi studerat. För vidare läsning om automatiserade tester rekommenderar vi *”Effecive Methods for Software Testing”* skriven av Perry, W. 1995. Vi är medvetna om att organisationer har väl utvecklade och avancerade tester som används därför har vi gjort en mer övergripande beskrivning av hur dessa tester kan gå till eftersom det hade varit övermäktigt att beskriva alla dessa tester. VOV-modellen uppmärksammar att tester är ett bra sätt att verfiera och validera system och ger en generell beskrivning av automatiserade tester.

Upptäcks fel under systemutvecklingen vid användandet av VOV-modellen ska man komma ihåg att uppdatera alla relevanta dokument för att utvecklingen ska vara kontrollerbar. Dokumentationens korrekthet är nödvändigt för att verifierings- och valideringsaktiviteterna ska genomföras på bästa möjliga sätt. Dokumentationen ger bra underlag för att tala om vart utvecklingsarbetet befinner sig och vilka milstolpar som uppnåtts och verifierats och validerats.

För att vinna gehör för kvalitetsstyrning och få resurser avsatta till VOV-modellen måste ledningen inse vikten av kvalitet. När val av utvecklingsmodell gjorts är det bra att bestämma vilka milstolpar som ska produceras under de olika utvecklingsfaserna.[69] Genom att använda VOV-modellen kvalitetssäkras de olika milstolparna allteftersom utvecklingen fortskrider och på detta sätt uppnås ett slutresultat som är av god kvalitet. Om de olika milstolparna som producerats under de olika utvecklingsfaserna bedöms vara komplexa ska dessa brytas ner till mindre stolpar, som VOV-modellen kallar delstolpar. Detta görs för att verifierings- och valideringsaktiviteterna ska kunna utföras på ett kontrollerbart sätt. Risken är annars att arbetet blir för stort och komplext vilket medför svårigheter att hålla allt under kontroll och få ett kvalitativt resultat. Efter verifiering och validering av de delstolpar, som tillsammans bildar en milstolpe, ska hela milstolpen verifieras och valideras innan systemets utvecklingsarbete fortsätter.

3.2 Verifierings- och valideringsaktiviteter i VOV-modellen

I följande avsnitt görs en beskrivning av hur VOV-modellens aktiviteter ska användas under analys- design- och implementeringsfasen. För att nå bästa resultat förs de

milstolparna eller delstolparna som produceras under de olika faserna in i VOV-modellen för verifiering och validering.

3.2.1 VOV-modellens analysfas

Under VOV-modellens analysfas utvecklas kravspecifikationen som är den milstolpe som ska verifieras och valideras. Kravspecifikationen är det dokument som ska dokumentera de krav som kunden har på det system som ska utvecklas. Det är därför viktigt att kravspecifikationen inte innehåller några fel och att den specificerar kundens krav korrekt. Om krav i kravspecifikationen är felaktiga kommer applikationen i ett senare skede att bli felaktig vilket ökar kostnadena (Se Ekonomiskt perpektiv kap. 2.1.1).

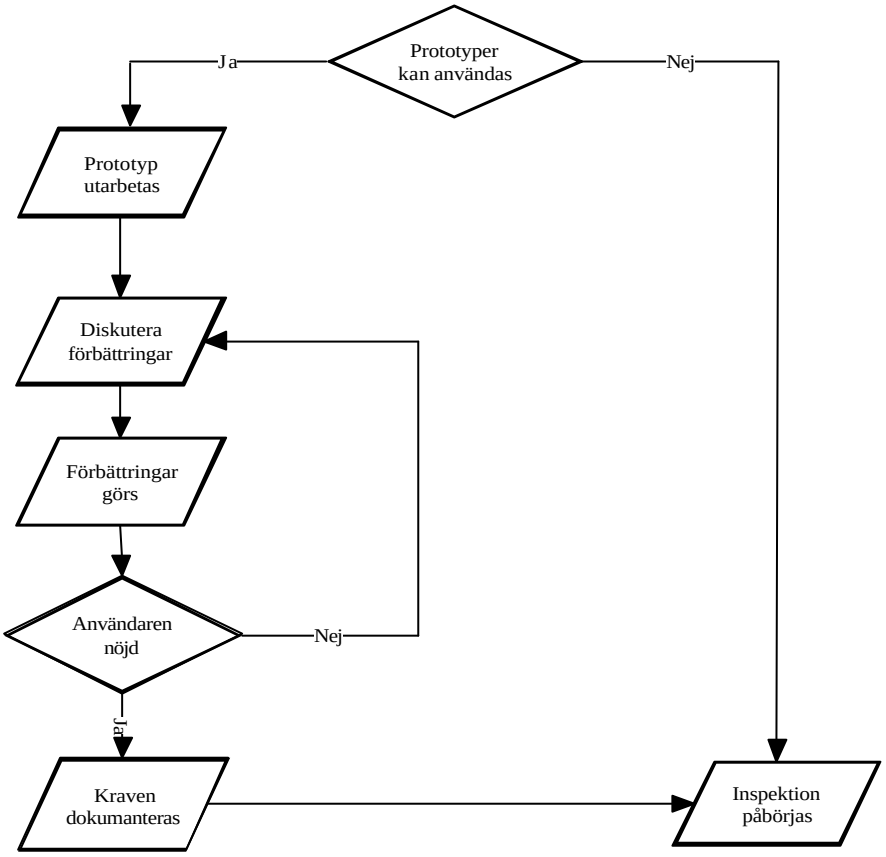
VOV-modellen stödjer dels analytisk systemutveckling där kravspecifikationen analyseras fram med hjälp av intellektuella bedömningar och dels experimentell systemutveckling där kravspecifikationen växer fram som ett resultat av experimentering med en konkret efterlikning av ett färdigt system, en prototyp. Att arbeta med prototyper är ett välkänt arbetssätt inom ingenjörsbranschen. Det innebär att en provversion görs av en kommande produkt innan man börja producera i stor skala. Prototypen behöver inte vara identisk med slutprodukten i alla avseenden. Prototypen används först och främst för att försäkra sig om att informationssystemet motsvarar användarnas önskemål och hjälper till att formulera användarnas krav. En prototyp är en modell av informationssystemet, som användarna själva kan utprova.[70]

Att formulera stabila och konsistenta krav är svårt. Det finns flera typer av inadekvata och ofullständiga kravspecifikationer. Ett fall är när kunden inte vet vilka krav systemet måste tillfredsställa och ett annat fall är när kunden vet vilka krav systemet måste tillfredsställa men vet inte hur dessa bör dokumenteras på ett fullständigt och tydligt sätt. Det är vanligt att krav modifieras och tillkommer under utvecklingens gång, allt eftersom kunden får en klarare syn på sina krav och inser att kraven som definierats från början i kravspecifikationen inte är fullständiga. Genom en nära kontakt med kunden minskas risken att kraven blir ofullständiga. Därför bör kravspecifikationen utformas i nära samarbete mellan systemutvecklare och kund och diskussionen angående kraven bör föras kontinuerligt under systemutvecklingen. För att kravspecifikationen ska bli lyckad krävs att systemutvecklaren och kunden etablerar en öppen och konstruktiv kommunikation där utvecklaren får förståelse för kundens verksamhet under analysfasen.

Verifiering och validering av kravspecifikationen utförs för att försäkra att kund och utvecklare har nått en överenskommelse angående vilka krav mjukvaran ska nå upp till och att man har samma syn på systemet innan man går vidare i systemutvecklings-processen. Det krävs att dokumentet skrivs på ett sätt som både användare och utvecklare förstår. Det kan vara användbart att ta sig tid att klara ut begrepp så att orden får samma betydelse för alla inblandade när man diskuterar kravspecifikationen, för att undvika missförstånd.

3.2.1.1 Verifiering och validering av krav med prototyper

Det huvudsakliga syftet med prototyper är att få en bra kravspecifikation.[71] Kravspecifikationen ska beskriva vilka önskemål användarna har på informationssystemet och följaktligen minska behovet av senare ändringar. Underhållskostnaderna för systemet kan följaktligen blir lägre.



Figur 6 En beskrivning av hur prototyper kan användas vid användandet av VOV-modellen.

En prototyp underlättar arbetet med att validera vissa av kraven. Prototypen behöver inte klara de belastningar som exempelvis många användare samtidigt, många transaktioner och stora datamängder, som systemet senare kommer att utsättas för. Det räcker att lösningen klarar en användare och några få data. Om prototypen innehåller användargränssnitt som blivit godkända av användaren så har gränssnitten validerats och ytterligare validering av dem behöver inte genomföras.

En prototyp är ett verklighetsnära exempel på de yttre egenskaperna hos det framtida systemet och prototypen innehåller de planerade skärmbilderna samt möjliggör den dialog mellan människa och maskin som man tror är lämpligast. Prototypen visar systemets funktionella egenskaper, det vill säga vad systemet kan göra för användaren och vad det kräver av användaren. Användarna, eventuellt flera efter varandra, provar sedan hur prototypen fungerar, och avgör om systemet uppfyller deras krav. Genom utprovningen kan användaren komma med nya önskemål utifrån vilka prototypen

modifieras. Processen upprepas tills man kommer fram till en situation som användarna trivs med.[72] Resultatet av prototypen betraktas som en del av användarnas kravspecifikation. Men eftersom alla krav inte är införda i prototypen behöver denna kompletteras med ytterligare krav som sedan ska verifieras och valideras vid en inspektion.

3.2.1.2 Verifiering och validering av krav med inspektioner

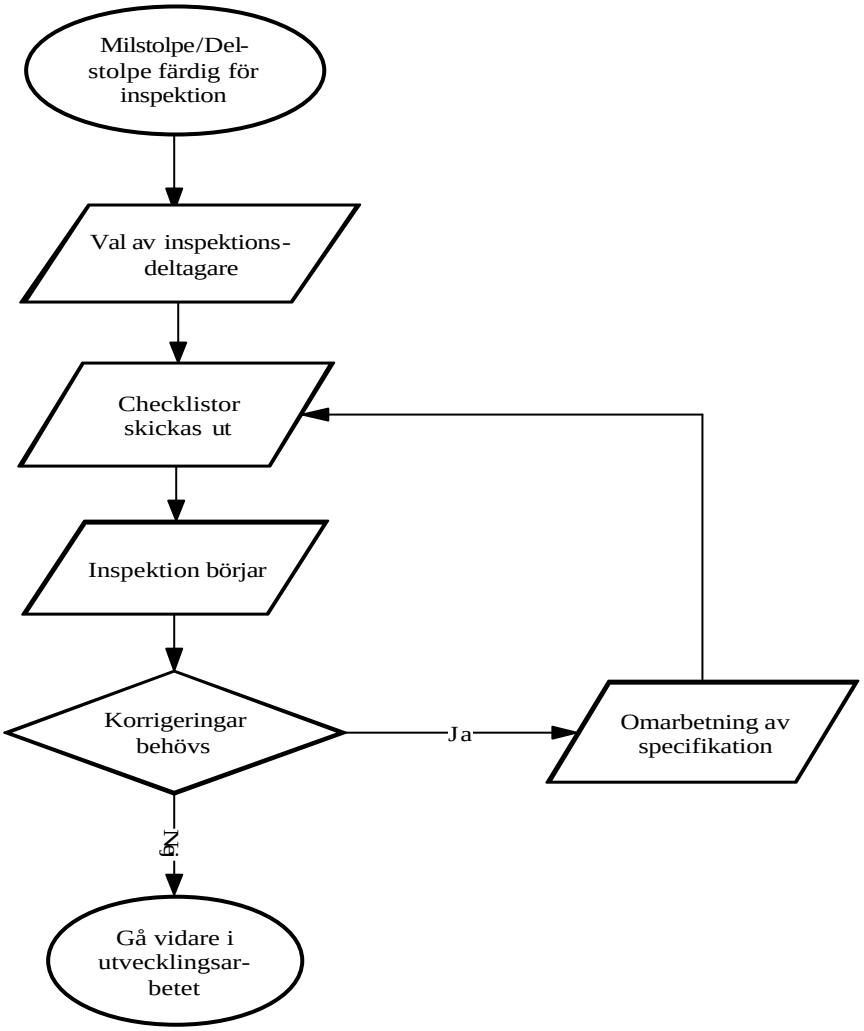
En inspektion av kravspecifikationen är en effektiv verifierings och valideringsaktivitet för att upptäcka felaktigheter i en kravspecifikation och innebär att den noggrant granskas av en grupp människor.[73] En inspektion är en manuell aktivitet som även används under design- och implementeringsfaserna i VOV-modellen. Eftersom kravspecifikationen ofta består av textuella dokument passar det bra att använda inspektioner för att verifiera och validera kravspecifikationen.

Inspektionerna måste schemaläggas och hanteras som viktiga delar i VOV-modellens analysfas. Om inspektionerna inte schemaläggs finns det risk för att aktiviteten hamnar i skymundan till fördel för det omedelbara arbetstrycket under systemutvecklingen. [74] Detta kortsiktiga sätt att tänka får negativa konsekvenser på lång sikt eftersom upptäckten och korrigering av fel skjuts upp till senare i utvecklingen, då kostnaden för att korrigera felet blir mycket större. (Se Ekonomiskt perspektiv kap. 2.1.1)

Inspektionen utförs av fyra till sex deltagare och fungerar bäst när deltagarna intar väl definierade roller. Deltagarna i en inspektion av kravspecifikationen bör ha god kännedom om kraven. Inspektionsdeltagarna ska vara författaren av kravspecifikationen, en person som är ledare för inspektionen, en person som ska utföra designen, en person som ansvarar för testningen, samt en person som representerar kunden. Det är viktigt att inspektionsgruppens sammansättning kompletterar varandra kunskapsmässigt och de olika deltagarna intar olika roller. Skulle det vara så att en person exempelvis både ansvarar för kravspecifikationen och är representant för kunden intar denne två av rollerna under inspektionen. Författaren av kravspecifikation ska vara närvarande p.g.a. att personen i fråga varit med och utformat kraven i dokumentet och därför förstår betydelsen av begreppen. Eftersom kravspecifikationen ska dokumentera de krav som kunden har på systemet sker inspektionen nödvändigtvis med en representant för kunden, som förstår slutanvändarens behov. I de fall när man varit många författare till ett och samma dokument ska en huvudansvarig författare utses som representant för krav-specifikationen. Inspektionsledaren är inspektionens nyckelperson och bör vara en kompetent programmerare som ska leda, schemalägga, skriva inspektionsrapporter samt svara för inspektionens uppföljning. Fördelen med att inspektionsledaren är en bra programmerare är att denne ser om ett krav är realistiskt genomförbart. Designern ska närvara vid inspektionen för att kontrollera att kraven är genomförbara i designfasen. Den person som representerar testningen ansvarar för att kraven inte är allt för subjektiva och utformas så att de blir testbara.

En inspektion bör inte vara längre än två timmar då gruppen efter denna tid blir mindre effektiv att hitta felaktigheter. Denna begränsning av tid anser vi vara bra då en

inspektion kräver koncentration och engagemang av sina deltagare och då vi av egna erfarenheter är medvetna om att koncentrationen tenderar att minska.



Figur 7 En beskrivning av hur en inspektion av kravspecifikationen går till.

En checklista ska användas under inspektionen för att identifiera inadekvata och ofullständiga krav. Med hjälp av checklistan fokuseras inspektionen på att deltagarna inte missar de vanligaste typerna av felaktigheter som kravspecifikationen kan innehålla. Den används även för att ge deltagarna en översikt av förhållanden som är viktiga att undersöka för att gardera sig mot dålig kvalitet på kravspecifikationen. Nedan görs en övergripande beskrivning av en checklistas innehåll som kan användas vid inspektionen. Utifrån denna checklista kan kontrollpunkter läggas till beroende på kravspecifikationens innehåll.

- Är alla krav *fullständiga*, tydliga och korrekta?
- Är kraven *konsistenta* med varandra?

- Är kraven *relevanta*, eller finns det onödiga krav?
- Är varje krav *möjligt att testa* för att bedöma om kravet uppnåtts?
- Kan varje krav spåras till kundens behov? Varför har kravet tagits med?
- Finns krav som preciserar *utvidgningsmöjligheter, säkerhet, åtkomst* och användarvänlighet specificerade?
- Är alla krav *möjliga att genomföra* utifrån givna resurser?
- Är formuleringen av kravet gjort på ett sådant sätt att det går att göra ändringar utan att det får konsekvenser för andra krav?
- Har kravspecifikationen dokumentarets på ett sådant sätt att den kan ligga till grund för vidare utvecklingsarbete?
- Beskriver kravspecifikationen systemets *koppling till andra system*?
- Finns *alla systemanvändare* beskrivna?
- Har man beskrivit hur det nya systemet kommer att påverka verksamheten organisatoriskt parallellt med systemutvecklingen?
- Är alla *funktioner* som kunden kräver specificerade på ett *tydligt och detaljerat* sätt?
- Är funktionernas *svarstider*, kapacitet och frekvens beskrivna?
- Är kraven på systemutvecklingens dokumentation beskrivna?

Innan en inspektion äger rum går varje deltagare, var och en för sig, igenom kravspecifikationen med hjälp av checklistan och markerar de delar de finner tvivelaktiga eller bedömer behöva vidare förklaring. Mötet börjar med att författaren av kravspecifikationen noggrant går igenom varje krav i dokumentet. Författaren förklarar kraven om någon vill ha en bättre beskrivning av dem och svarar på frågor. För att underlätta beskrivningen av kravspecifikationen kan hjälpmedel som white-board och post-it lappar användas. Därefter går deltagarna en och en igenom sin lista med markerade krav som är oklara eller felaktiga. Ur denna diskussion som uppstår avslöjas felaktigheter. Om inspektionen visar att kravspecifikationen innehåller felaktigheter noteras dessa i en inspektionsrapport. Exempel på felaktiga krav kan vara orealistiska, motsägelsefulla, glömda eller ej testbara krav. Felaktigheterna ska sedan omarbetas och inspektionsledaren följer upp att kraven rättats till på ett korrekt sätt. Inspektionsledaren bedömer om omarbetningarna är av god kvalitet och om man behöver ytterligare en inspektion på de krav som omarbetats eller om en ny inspektion ska utföras på hela kravspecifikationen. Anledningen till att hålla en ny inspektion på hela kravspecifikationen är att ändringar av krav kan medföra att andra krav påverkas och blir felaktiga. Genom verifiering och validering av kravspecifikationen kan man upptäcka inadekvata krav som kanske annars inte skulle upptäckts lika tidigt i systemutvecklingen.

När inspektionsledaren beslutat att kravspecifikationen är fullständig och ej behöver verifieras eller valideras ytterligare ska utvecklingen gå vidare enligt den utvecklingsmetod som används. Enligt VOV-modellen kan systemutvecklaren när som helst under systemutvecklingen gå tillbaka och utföra ytterligare verifierings- och valideringsaktiviteter på kravspecifikationen om förändringar gjorts i kravspecifikationen vid någon annan fas under systemutvecklingen. När en förändring i kravspecifikationen gjorts är det viktigt att verifiera och validera denna på nytt genom inspektioner.

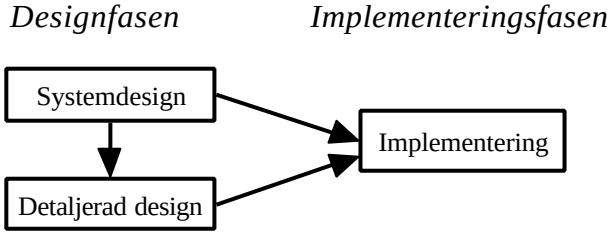
3.2.2 VOV-modellens designfas

Designfasen involverar ett nära samarbete mellan kund och utvecklare, för att systemet ska utvecklas till att motsvara kundens förväntningar. Resultatet av designfasen ska, liksom resultatet av andra faser i utvecklingsprocessen, verifieras och valideras innan utvecklingsarbetet går vidare. Det är mycket eftertraktat att hitta fel i designfasen då senare avlägsnande av felet kan bli mycket dyrare (se figur 3).

För att kunden ska kunna läsa, förstå och kritisera designen krävs det att den är utformad på ett klart och enkelt sätt, skriven med kundens språk och ritade i klara modeller och diagram. De tekniska delarna av designen behöver inte förstås av kunden men de ska vara lätta för en utvecklare att läsa och förstå. Designen ska vara sådan att användarna kan förstå alla icke tekniska delar. Detta är en skillnad jämfört med kravspecifikationen där både kund och utvecklare ska kunna läsa och förstå hela dokumentationen. Ett sätt är att bryta ner designen inkrementellt till hanterbara delar. På detta sätt kan användarna förstå och se om delarna är korrekta samt föra sina åsikter vidare till utvecklaren som sedan kan modifiera designen.

Designaktiviteter av ett system delas in i två nivåer. I den första nivån skapas systemdesignen där de komponenter som är nödvändiga i systemet definieras. Den andra nivån producerar en detaljerad design som förfinar systemdesignen genom att erbjuda mer logiska beskrivningar hos komponenter och datastrukturer.[75]

Den detaljerade designen ska medföra att det blir lättare att implementera designen i kod, men det är inte nödvändigt att lägga in hela systemdesignen i en detaljerad design. Implementering av kravspecifikationen kan utföras utan den detaljerade designen.



Figur 8 En beskrivning av hur designen kan utföras.

3.2.2.1 Verifiering och validering av systemdesign

Systemdesignen utformas utifrån kravspecifikationen och kommer senare lägga grunden till implementering av systemet. Systemdesignen ska verifieras och valideras för att försäkra att designen implementerar alla krav som ställts i kravspecifikationen, kontrollera att kunden är nöjd samt om designen kommer att kunna översättas till effektiv kod under implementeringen. Designen ska även granskas för att avgöra om det är möjligt att modifiera existerande funktioner och om nya funktioner kan läggas till. Alla komponenter i systemdesignen ska sträva efter att vara löst kopplade,[76] vilket innebär att komponenterna kan betraktas som oberoende av varandra och lättare kan anpassas efter en designförändring.

I VOV-modellen används genomgångar och inspektioner för att verifiera och validera systemdesignen. Målet med genomgångarna och inspektionerna är att försäkra sig om att systemdesignen tillfredsställer kravspecifikationen och är av god kvalitet. Genomgångar används när en del av systemdesignen producerats och inspektioner då en större del av systemdesignen utformats.

Genomgångar kan vara informella eller formella möten som sammankallas på förfrågan av en författare till en del av systemdesignen där denne ber någon eller några att ha en genomgång. Under genomgången granskas systemdesignen. Då en genomgång kan vara informell krävs inte att några dokument av mötet produceras. Det kan dock vara fördelaktigt att dokumentera felaktigheter som upptäcks under en genomgång så att man kan gå tillbaka i dokumenten för att förstå varför vissa designbeslut fattades. Anledningen till att en genomgång väljs istället för en inspektion kan vara att inspektionen tar fler personers tid och kraft i anspråk och det borde vara allt för dyrt att verifiera och validera varje delstolpe vid inspektioner. Istället räcker det att granska systemdesignen genom inspektioner när större delar av systemdesignen producerats.

Inspektioner av systemdesignen fungerar på liknande sätt som inspektioner av krav-specifikationen. De ska schemaläggas och planeras av inspektionsledaren. De personer som deltar i inspektionen ska vara designern till systemdesignen, designern till den detaljerade designen, en person som ska implementera systemet med kod, författaren till kravspecifikationen samt en inspektionsledare. Deltagarna har innan mötet fått ut systemdesignen och en checklista som vägledning för att förbereda sig inför inspektionen. Var och en markerar oklarheter och felaktigheter som de upptäcker. Inspektionen börjar med att systemdesignern förklarar designen där varje logisk del granskas minst en gång för att deltagarna ska förstå designen. När designen gåtts igenom ställer de övriga deltagarna frågor. Den diskussion som uppstår leder till att fel hittas i systemdesignen. Under mötet ska all tidigare dokumentation finnas tillgänglig för deltagarna, ifall man har behov av att gå tillbaka i dokumentationen. Checklistan används som hjälpmedel vid inspektionen och kan ha följande innehåll och kan vid behov kompletteras med ytterligare kontrollpunkter:

- Finns alla funktionella krav designade?
- Har man *missat* några *krav*?
- Har man beskrivit hur *testningen* av varje krav ska utföras?
- Har stora *designbeslut dokumenterats*?
- Är designen konstruerad i enlighet med utvecklingsmodellens *designstandard*?
- Inbegriper designen saker som *felhantering, användarvänlighet och utbyggbarhet*?
- Hanteras alla *exceptionella händelser*?
- Har *gränssnitten* beskrivits på ett riktigt sätt?
- Är designens *komponenter oberoende* av varandra?
- Har man uppskattat *storleken på systemet*?

Inspektionsledaren ska föra in alla systemdesignfel som upptäcks under inspektionen i en rapport och sedan följa upp att dessa omarbetats på ett kvalitativt sätt. Inspektionsledaren

avgör om det omarbetade materialet ska inspekteras i ännu en inspektion, eller om det räcker med en genomgång.

När verifiering och validering av systemdesignen gjorts finns det vissa delar av designen som är fördelaktiga att föra in i en detaljerad design. Hela systemet ska inte läggas in i en detaljerad design utan endast de delar som för systemet är komplexa och viktiga moduler. Hur mycket detaljerad design som behöver skrivas beror på vilken erfarenhet utvecklarna besitter och på projektets svårighetsgrad. Om utvecklarna har mycket erfarenhet av systemutveckling och om projektet är relativt enkelt kan den detaljerade designen vara informell och ske parallellt med konstruktionen. Man kan då se den detaljerade designen som en informell process där programmeraren förbereder designen som ett steg mot kodning, utan att dokumentera detta i ett formellt dokument. Jämfört med krav-specifikationen och systemdesignen har dokumentationen av den detaljerade designen mindre värde för senare underhåll av systemet. Detta beror på att det är svårt att hålla den detaljerade designen uppdaterad allt eftersom kodningen fortskrider.[77] En logisk förklaring av systemet som ändå inte beskriver systemet korrekt är ingen hjälp för kodningen. De delar som förts in i en detaljerad design verifieras genom att antingen utföra genomgångar eller inspektioner enligt VOV-modellen.

3.2.2.2 Verifiering och validering av detaljerad design

Verifiering och validering utförs för att kontrollera att den detaljerade designen möter specifikationen som lagts ner i systemdesignen. Verifiering och validering av den detaljerade designen ska försäkra att den svarar mot alla relevanta krav, på motsvarande sätt som systemdesignen gör, och se till att den är konsistent med systemdesignen. Vidare kontrolleras om alla krav som specificerats går att följa genom den detaljerade designen. Vid verifiering och validering av den detaljerade designen fokuserar systemutvecklare på hur systemet är organiserat på enhets och rutinnivå, till skillnad från granskning av systemdesign där de fokuserar på programmets systemnivå.[78]

Verifiering och validering av den detaljerade designen sker vid genomgångar och inspektioner, där man manuellt går igenom och kritiskt granskar designen. En genomgång av den detaljerade designen görs antingen vid ett formellt eller informellt möte som sammankallas av designern där han ber någon eller några att ha en genomgång, på samma sätt som genomgång av systemdesignen. Genomgången innebär en aktivitet där designern logiskt steg för steg förklarar den detaljerade designen och de som är med på mötet ställer frågor, pekar på möjliga felaktigheter och ber designern klarifiera delar av designen som verkar oklar. En sidoeffekt av mötet kan vara att designern själv kan komma på fel och brister i designen efter det att designen i detalj gåtts igenom. Eftersom genomgången kan vara informell så krävs det inte att den dokumenteras men de ändringar som görs under genomgången ska uppdateras i alla relevanta dokument.

Inspektionen av den detaljerade designen är en formell aktivitet som utförs på samma sätt som vid inspektionen av systemdesignen och ska schemaläggas. Inspektionen går ut på att försäkra att den detaljerade designen tillfredsställer specifikationen som lagts i systemdesignen. Deltagarna i inspektionen är de personer som deltagit vid inspektionen

av systemdesignen. Checklistan som används vid inspektionen av den detaljerade designen har dock ett annorlunda innehåll än vid systemdesignen. Exempel på checklistans innehåll beskrivs med följande punkter och kan kompletteras med egna kontrollpunkter:

- Existerar alla *moduler* som finns i systemdesignen även i den detaljerade designen?
- Har man tänkt igenom hur man ska bevisa att varje krav *uppnåtts*?
- Reflekteras alla relevanta aspekter av systemdesignen även i den detaljerade designen?
- Hanteras alla *exceptionella händelser*?
- Stämmer alla *format* överens med systemdesignen?
- Har man uppskattat *storleken på systemet*?
- Kan alla *tillstånd* som beskrivits i vanligt språk lätt *översättas till kod*?
- Är alla *villkor* korrekt *specificerade* och *accepterbara*?
- Har *uppbyggnaden* av designen blivit allt *för komplex*?
- Kan designens komponenter betraktas som *självständiga delar*?

Korrigeringar av designen görs efter inspektionen av designern och följs sedan upp av inspektionsledaren som bedömer om inspektionen ska upprepas.

När verifiering och validering av både system- och detaljerad design har genomförts och följts upp av inspektionsledaren kan utvecklingsarbetet fortsätta enligt den utvecklingsmetod som används. Vid systemutveckling kan ett fel införas när som helst i utvecklingen och det är därför viktigt att kontinuerligt granska de dokument som produceras. De fel som fortfarande kvarstår efter verifiering och validering under analys- och designfaserna implementeras i koden samtidigt som fler fel kan uppstå i kodningsfasen. Utifrån denna aspekt ingår därför verifierings- och valideringsaktiviteter under implementeringsfasen i VOV-modellen.

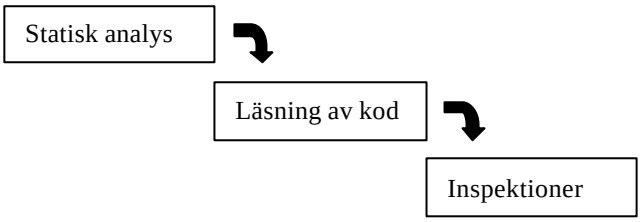
3.2.3 VOV-modellens implementeringsfas

I implementeringsfasen översätts systemdesignen och den detaljerade designen till kod. Målet med verifiering och validering av koden som producerats är att försäkra sig om att koden är konsistent med designen och kravspecifikationen samt att försäkra sig om att systemet är rätt för kunden. Vid verifiering och validering av kod används både manuella genomgångar och automatiserade tester. Vid manuella genomgångar genomförs ingen exekvering av programmet, vilket istället görs vid de automatiserade testerna. De former av manuella genomgångar som ingår i VOV-modellen är statisk analys, läsning av kod och kodinspektioner. Genom dessa aktiviteter lokaliseras oftast felen direkt till skillnad från automatiserade tester där endast förekomsten av fel upptäcks utan att för den skull veta vart felet ligger. Vid automatiserade tester exekveras testdata i systemet och utmatningen från systemet kontrolleras för att upptäcka felaktigheter. De typer av fel som upptäcks genom manuella genomgångar är ofta inte av samma typ som de fel som upptäcks genom testning. Testning och manuella genomgångar kompletterar varandra och båda ska användas för att få ett tillförlitligt system.[79]

Nedan görs en beskrivning av de olika verifierings- och valideringsaktiviteter som används i VOV-modellen för att kontrollera koden. Först ges en beskrivning av de manuella genomgångarna och sedan ges en beskrivning av de automatiserade testerna.

3.2.3.1 Verifiering och validering med manuella genomgångar

De manuella genomgångar som används i VOV-modellen är statisk analys, läsning av kod och inspektioner. Innan inspektioner av kod genomförs ska statisk analys och läsning av kod utföras.



Figur 9 Verifiering och validering genom manuella genomgångar.

3.2.3.1.1 Statisk analys

Vid statisk analys verifieras koden genom att metodiskt analysera programkoden. Statisk analys görs mekaniskt med hjälpmedel från programverktyg. Under statisk analys exekveras inte programmet, men programkoden används som inmatning till program-verktyget. Ett exempel på ett programvaruverktyg som utför begränsad statisk analys är en kompilator. Under statisk analys hittas avvikelser som t.ex. oanvända variabler. Genom att utföra statisk analys på koden minskas arbetet vid testningen. [80]

3.2.3.1.2 Läsning av kod

Ytterligare ett sätt att verifiera koden manuellt är att programmeraren läser koden för att upptäcka eventuella avvikelser mellan designspecifikationen och implementationen. Vid läsning av kod börjar man från detaljerna i systemet och går sedan vidare till en mer abstrakt beskrivning. Denna process är helt tvärtom om man jämför med designfasen. Vid designfasen börjar man från en abstrakt nivå och går mot mer en detalj nivå. Läsning av kod är ett användbart sätt att upptäcka fel som ofta inte upptäcks genom testning. [81]

3.2.3.1.3 Inspektion av kod

Inspektion av kod utförs efter det att andra former av manuella genomgångar har använts men innan testningen påbörjas. Därför ska defekterna som hittas vid statisk analys och läsning av kod korrigeras innan inspektion äger rum.

Deltagarna vid inspektionen ska vara duktiga programmerare och systemutvecklare. Inspektionsledaren och den designer som var med vid designinspektionen ska vara närvarande eftersom dessa har förståelse för hur man tänkt vid utvecklingen av designen. De som varit ansvariga vid implementeringen av de olika delarna av koden som ska inspekteras, ska deltaga. För att vara på den säkra sidan om att implementeringen är konsistent med kravspecifikationen ska även den person som ansvarar för krav-specifikationen vara närvarande. Vid inspektionen av kod ska deltagarna i förväg granska koden med hjälp av en checklista för att upptäcka eventuella felaktigheter. Under

inspektionen förklarar de som skrivit koden upplägget och ur den diskussion som uppstår försöker man finna defekter och samtidigt föreslår deltagarna möjliga anledningar till varför defekten uppstått.

Checklistans innehåll varierar beroende på vilket programmeringsspråk som används vid implementeringen. Den checklista som används vid kodinspektionen kan ha följande kontrollpunkter men bör kompletteras för att göra checklistan konsistent med programmeringsspråket man använder:

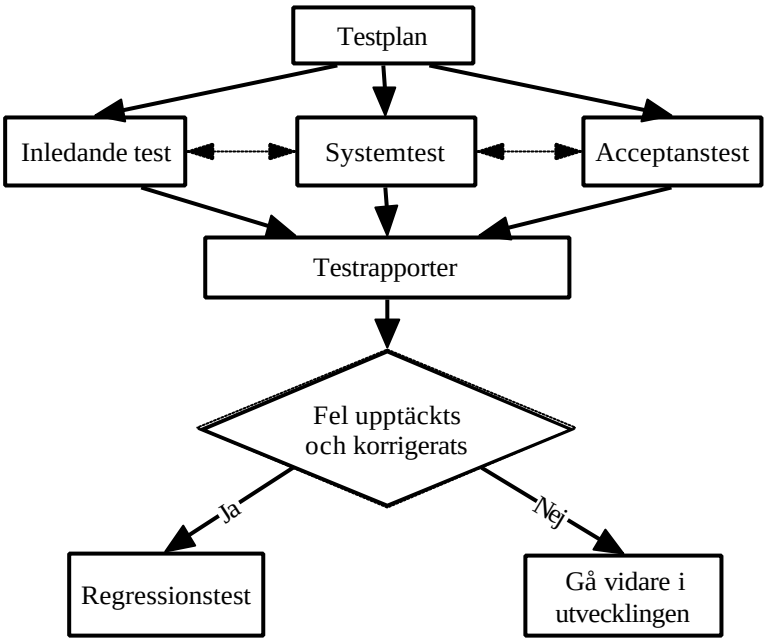
- Är designen *implementerad fullständigt* och korrekt?
- *Saknas* det några *externa funktioner*?
- Är alla *villkor korrekta*? Loopar, case- och if –satser?
- Finns det någon *kod* som är *onödig*?
- Är koden *lättläst*? Logiska variabelnamn, användandet av kommentarer etc.
- Har *testbarhet* byggts in i koden?
- Är koden *strukturerad* på ett sätt så att *nya funktioner* lätt kan föras in i koden?
- Används *felhantering* på ett meningsfullt sätt?

Inspektionsledaren är den person som ansvarar för att felaktigheterna förs in i inspektionsrapporten och följer sedan upp att dessa har åtgärdats. När de manuella genomgångarna för att verifiera koden har utförts tar de automatiserade tester över verifierings- och valideringsaktiviteterna.

3.2.3.2 Verifiering och validering med automatiserade tester

Testning används för verifiering och validering av kod, där de delar av programmet som ska testas exekveras och systembeteendet observeras. Genom observationen upptäcks felaktigheter i systemet och förekomsten av fel kan minskas genom att senare avlägsna felet. Testning är traditionellt en dyr aktivitet med tanke på att fel som upptäcks sent i utvecklingen är dyra att korrigera. Men har man lagt ner tyngd på testbarhet i systemutvecklingen kan man snabbare, enklare och billigare erhålla god pålitlighet hos systemet. För att minska kostnaderna för testning måste man ha en kvalitativ och välorganiserad inställning till systemutveckling. För att genomföra effektiva test krävs noggrann planering av testningen. VOV-modellen uppmärksammar att testbarhet av krav byggs in i systemet på ett tidigt stadium vilket underlättar testningen.

Testning bevisar endast att programmet innehåller fel. Syftet med testning är att hitta fel i koden och alltså visa att något är inkorrekt.[82] Under testning exekveras de delar av programmet som ska testas med ett antal testfall där man sedan utvärderar utdata efter exekveringen för att avgöra om programmet fungerar som man tänkt sig.



Figur 10 Den automatiserade testprocessen.

Att testa ett stort system är en komplex aktivitet och måste brytas ned i mindre aktiviteter, där komponenter och delsystem i systemet testas separat innan de integreras till ett system inför systemtestningen.[83]

3.2.3.2.1 Testplan

Testningsprocessen påbörjas med en testplan, som är det grundläggande dokumentet som guidar hela testningen av systemet. En testplan innehåller beskrivningar som t.ex. testprocessen, komponenter som ska testas, dokumentation av de utförda testerna och en bra plan för verifiering av att kraven har uppfyllts. Testplanen specificerar nivåerna på testningen och de enheter som behövs testas. För var och en av de olika enheterna, specificeras först testfallet och sedan granskas de. Under testet exekveras testfallet och testrapporter skapas för att utvärdera testningen.[84]

3.2.3.2.2 Teststrategier

Det finns flera olika teststrategier vars användbarhet beror på systemdesignens utseende. Vid större system där klara skillnader i strukturer mellan delsystemen finns, används olika teststrategier för test av de olika delsystemen. Valet av teststrategi påverkas av hur projektgruppen ser ut och vilken strategi projektmedlemmarna har mest erfarenhet av. Oberoende av strategival, är det ändå viktigt att använda sig av någon form av inkrementell testordning. Det innebär att börja någonstans i strukturen och sedan testa sig

igenom denna steg för steg. Fel som upptäcks i någon fas kan då direkt åtgärdas, vilket inte hade gått om man t.ex. hade slagit ihop alla enheter på en gång och testat helheten.[85]

3.2.3.2.3 Inledande test

Eftersom testning är beroende av den utvecklingsmodell och det programmeringsspråk som används vid systemutvecklingen så specificeras inte inledande tester i VOV-modellen. De komponenter, delsystem och enheter som implementerats ska testas självständigt innan man sätter ihop och testar hela systemet med systemtest. Sammansättningen av ett system bör ske stegvis där enheter sätts ihop till delsystem och testas var för sig innan de olika delsystemen förs samman och testas. På detta sätt lokaliseras felen tidigare vid testningen och det blir lättare att korrigera felen.[86]

3.2.3.2.4 Systemtest

Delsystemen integreras och bildar det färdiga systemet. Syftet med detta test är delvis att kontrollera att samarbetet mellan delsystemen och andra delar av programmet verkligen fungerar som det var avsett från början. I denna fas ingår också att kontrollera systemet mot dess funktionella och icke-funktionella krav.[87]

3.2.3.2.5 Acceptanstest

Acceptanstest är en valideringsaktivitet och är den sista fasen i testprocessen innan mjukvaran anses färdig. Detta test involverar kunden, som är med vid testkörningen. Testet ska avslöja om systemet uppfyller de krav som finns i kravspecifikationen. Även andra avslöjanden som t.ex. onödiga eller saknade krav kan visa sig vid acceptanstest. Men har VOV-modellen använts under hela systemutvecklingen ska det ej finnas saknade eller onödiga krav, då systemutvecklare bibehållit kommunikationen med kunden under hela utvecklingsarbetet. Acceptanstestet pågår tills systemutvecklare och kund kommer överens om att det färdiga systemet fungerar enligt kravspecifikationen. En skillnad mellan systemtest och acceptanstest är att vid systemtest simuleras ofta omgivningen (hårdvaran, andra mjukvaror m.m.), medan acceptanstestet utförs på den verkliga omgivningen.[88]

3.2.3.2.6 Regressionstest

När någon del i koden modifieras måste de delar av systemet som påverkats av ändringen testas åter igen. Avsikten med regressionstestet är att försäkra sig om att systemet fortfarande uppfyller kraven. När ett fel upptäcks under någon testfas måste detta givetvis korrigeras. För att vara säker på att korrigeringen inte introducerade något nytt fel, måste testprocessen upprepas. Denna testprocess är iterativ och i princip skall alla testfaser upprepas på hela systemet efter varje korrigering. I praktiken är nog detta väldigt dyrt så därför ska man i VOV-modellen istället testa en delmängd av systemkomponenterna.[89]

3.2.4 Sammandrag av VOV-modellens utformning

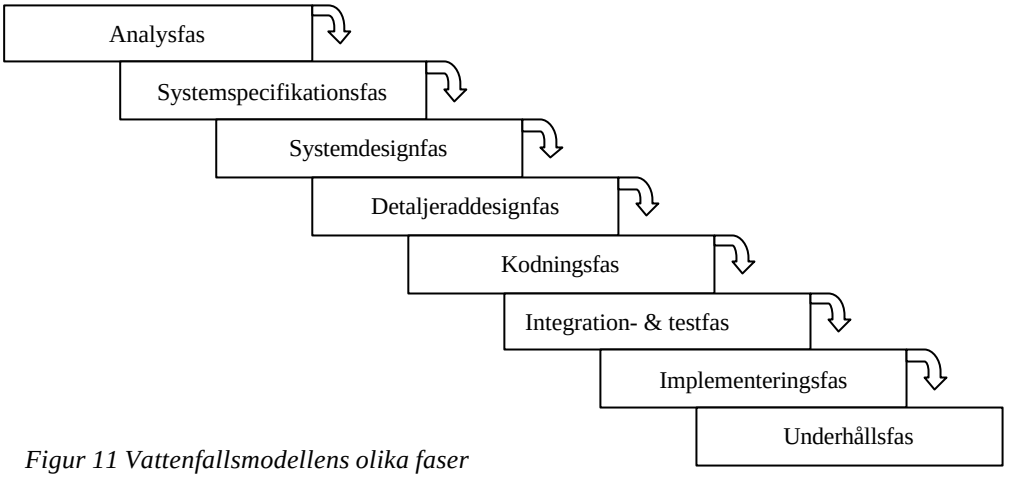
Genom att utföra de verifierings och valideringsaktiviteter som beskrivits ovan i VOV-modellen som komplement till den systemutvecklingsmetod man använder höjer man systemets kvalitet. Detta eftersom de milstolpar och delstolpar som produceras under utvecklingen kontinuerligt verifieras och valideras vilket kvalitetssäkrar systemet. Används VOV-modellen fokuseras verifierings- och valideringsaktiviteterna inte endast under implementeringsfasen genom testning utan även under analys- och designfaserna och fel som tidigt introducerats i systemet avlägsnas. Dessutom uppmärksammas de olika typer av fel som vanligtvis förknippas med respektive utvecklingsfas så att kontroller kan utföras för att undvika och förebygga felen.

4. ANVÄNDNING AV VOV- MODELLEN

I detta kapitel beskriver vi först hur det går till när system som utvecklas med vattenfalls-modellen kompletteras med VOV-modellen. Därefter beskriver vi på liknande sätt hur systemuvecklingen ser ut om en objektorienterad metod kompletteras med VOV-modellens verifierings och valideringsaktiviteter. Slutligen gör vi en jämförelse mellan de två olika sätten att systemutveckla och beskriver vilka likheter respektive skillnader vi finner samt vilka för- och nackdelar som visar sig.

4.1 Systemutveckling med vattenfallsmodellen och VOV-modellen

Att välja vilken form av utvecklingsmetod som ska användas när ett system ska utvecklas kan vara en svår uppgift. Den enklaste utvecklingsmetoden är vattenfallsmodellen där faserna är organiserade i en linjär ordning. Det finns flera olika variationer av vattenfallsmodellen beroende på aktiviteterna och flödet av kontroller mellan dem. Vi har valt att arbeta med den traditionella vattenfallsmodellen, vilket innebär att inget iterativt arbete utförs under utvecklingen. För att det ska bli ett framgångsrikt projektresultat och ett framgångsrikt system ska alla faserna i vattenfallsmodellen utföras och detta i en bestämd ordning. Om en annan ordning på faserna används kommer detta att resultera i ett mindre framgångsrikt system.[90]



Figur 11 Vattenfallsmodellens olika faser

Inga tillbakagångar är möjliga då man inte kan klättra tillbaka uppför vattenfallet. För att klart och tydligt identifiera slutet på en fas måste man specificera vilka aktiviteter som ska utföras och vilka produkter som ska skapas innan nästa fas kan påbörjas. När aktiviteterna i en fas genomförts ska detta resultera i en produkt som sedan granskas för att försäkra att produkten dels är konsistent med dess ”inmatning” och dels att den är konsistent med kraven i systemet.[91]

Vattenfallsmodellen börjar med att undersöka projektets genomförbarhet och samtidigt utvärderas inställningen för utformning och utveckling hos verksamheten. Detta görs med

den kravspecifikation som man får av verksamheten, vilken anser sig behöva utveckla sitt system eller skaffa sig ett nytt.[92]

Nästa steg i modellen är en systemspecifikationsanalys för att bestämma vad systemet ska klara av, dvs kraven på produkten och detta gör man genom en systemkravspecifikation. Med systemkravspecifikationen som grund går modellen in i produktdesignen, där en specifikation av arkitekturen och designen på produkten utarbetas vilket resulterar i systemdesigndokument. Nästa steg i modellen behandlar även design, men mer detaljerat, där arbete läggs på att göra en detaljerad design över projektets moduler, vilket resulterar i en detaljerad design.[93]

Det fjärde steget i vattenfallsmodellen är kodning, där programmering och enhetstester av varje modul utförs. Efter detta steg integreras och testas systemet med testplaner och testrapporter. Vid implementeringssteget installeras och integreras mjukvaran i den verkliga miljön och samtidigt skall det finnas programmanualer som användar- och installationsmanualer tillgängliga. Det sista steget i vattenfallsmodellen behandlar underhållet av systemet som utvecklats, där en kontroll utförs och uppdaterar systemet ifall det skulle behövas.[94]

Följande produkter kan ses som ett minimum som borde produceras i varje projekt som utvecklas med vattenfallsmodellen.[95]

- Kravspecifikation
- Systemkravspecifikation
- Systemdesigndokument
- Detaljerad designdokument
- Testplaner och testrapporter
- Slutlig programkod
- Programvarumanualer (T.ex. användar- och installationsmanualer)

När system utvecklas med vattenfallsmodellen och med VOV-modellen som komplement betraktas ovanstående produkter som milstolpar vilka ska verifieras och valideras. Verifiering och validering av ett system som utvecklas enligt vattenfallsmodellen sker i slutet av varje systemutvecklingsfas, under hela utvecklingsprocessen. Men med VOV-modellen som komplement är det även möjligt att bryta ner milstolparna till delstolpar för att verifiera och validera dessa även under varje systemutvecklingsfas. Alltså inte bara i slutet av varje fas.

4.1.1 Vattenfallsmodellens analys

Analysfasen i VOV-modellen berör vattenfallsmodellens analys- och system-specifikationsfaser. Under vattenfallsmodellens analysfas skapas en kravspecifikation och i systemspecifikationsfasen utvecklas en systemkravspecifikation och om dessa specifikationer behöver kompletteras för att tydliggöra kraven används även olika typer av modeller och diagram.[96]

4.1.1.1 Kravspecifikation

Vid analysfasen i vattenfallsmodellen görs en förändringsanalys över den verksamhet, som vill utvecklas eller skaffa ett nytt system, utifrån en specifikation som kunden skrivit. Förändringsanalysen görs dels för att undersöka och validera om det är genomförbart att utveckla systemet som kunden vill ha med bl.a. tanke på resurser, ekonomi och om kraven som är ställda är realiserbara och dels för att få fram eventuella utvecklingsåtgärder. Kraven utvecklas sedan i kommunikation mellan systemutvecklare och kund genom både informella och formella aktiviteter. Kravspecifikationen ska vara lätt att läsa och förstå av både kund och utvecklare. Enligt vattenfallsmodellen måste kravspecifikationen vara fullständig innan systemutvecklingen kan gå vidare.

4.1.1.2 Systemkravspecifikation

För att utvecklingen av systemet ska vara möjligt måste kravspecifikationen skrivas med ett låg-nivå språk i en systemkravspecifikation, så att den blir användbar för systemutvecklingen. Systemkravspecifikationen är det dokument som ska innehålla en fullständig specificering av vilka krav systemet ska implementera. Denna specifikation är en abstrakt funktionell beskrivning men blandar in implementering så lite som möjligt för att dokumentet ska bli så klart och precist som möjligt. Det är svårt att producera en systemkravspecifikation som är skriven med ett tillräckligt högt nivåspråk som är läsbart samtidigt som den är skriven med ett tillräckligt lågt nivåspråk för att fullständigt och precist definiera systemets mjukvarukrav.[97]

Kraven i kravspecifikationen utvärderas och analyseras under vattenfallsmodellens systemkravspecifikation för att förbättra och utveckla kraven i kravspecifikationen. Detta måste göras för att systemutvecklaren mer specifikt ska veta vad kunden vill ha. Exempel på ett krav som ofta förekommer i kravspecifikationen är att ”systemet måste vara användarvänligt”. Detta krav måste självklart omdefinieras i mer mätbara termer som har betydelse för systemutvecklare, t.ex. ”en otränad användare måste kunna utföra vissa funktioner som systemet tillgodoser inom en viss tid.”[98] När systemkravspecifikationen har utvecklas ska systemutvecklarna ha en fullständig beskrivning över kraven som systemet ska implementera. Enligt vattenfallsmodellen måste systemkravspecifikationen vara färdig innan designen av systemet kan påbörjas.[99]

4.1.1.3 Verifiering och validering under VOV-modellens analysfas

VOV-modellen inkluderar både vattenfallsmodellens analys- och systemspecifikationsfas i sin analysfas. De milstolpar som producerats under dessa två faser är dels en krav-specifikation och dels en systemkravspecifikationen. De eventuella diagram och modeller som använts för att förtydliga vissa delar i specifikationerna betraktas som delstolpar. Enligt vattenfallsmodellen så ska kravspecifikationen valideras innan system-specifikationsfasen kan påbörjas och systemkravspecifikationen måste valideras innan systemspecifikationsfasen kan avslutas.[100]

Om vattenfallsmodellen kompletteras med VOV-modellen leder detta till att de specifikationer som producerats inte bara ska valideras utan även verifieras för att kontrollera att specifikationerna är konsistenta med varandra och att de är uppbyggda på rätt sätt. De båda specifikationerna som producerats ska granskas genom inspektioner på det sätt som beskrivs i VOV-modellen och på så sätt kvalitetssäkras specifikationerna. De modeller och diagram som används för att tydliggöra kraven ska inkluderas i inspektionerna och granskas med lika stor uppmärksamhet som milstolparna. Genom att använda VOV-modellen kan verifiering och validering av del- och milstolparna utföras, inte bara då de anses vara färdiga, utan även inspekteras under själva konstruktionen.

Att arbeta med prototyper förknippas vanligtvis med iterativt arbete mellan olika systemutvecklingsfaser, något som inte tillåts enligt vattenfallsmodellen. Men arbetet med prototyper enligt VOV-modellen innebär att prototypen utarbetas endast under analysfasen. Om VOV-modellen används som komplement så är det möjligt att göra en prototyp även när ett system utvecklas enligt vattenfallsmodellen. Prototypen kompletteras sedan med krav- och systemkravspecifikationer enligt vattenfallsmodellen.

4.1.2 Vattenfallsmodellens design

Designfasen i VOV-modellen inbegriper vattenfallsmodellens systemdesignfas och detaljerad designfas. De milstolpar som produceras är en systemdesign och en detaljerad design.

I vattenfallsmodellen måste systemskravspecifikationen vara fullständig för att systemutvecklarna ska kunna börja med design av systemet. I designfasen definieras systemets arkitektur och målet med designfasen är att reducera abstraktionsnivån och beskriva hur systemet ska implementeras enligt systemkravspecifikationen. Designen beskriver hur olika data ska representeras och olika konstruktioner bryts ner och struktureras i mer detaljerade beskrivningar. Först designas systemdesignen och sedan utformas en detaljerad design. Designen utvecklas stegvis där allt mer detaljer introduceras i varje steg, designen går från en hög nivå till att bli på en allt lägre nivå.[101]

Syftet med designfasen är att kartlägga en lösning av det problem som specificerats i systemkravspecifikationen. Denna fas är det första steget att gå från problem till lösning. I början av utvecklingen definieras vad som behövs och när designen tar sin fart visar den hur dessa behov ska tillfredsställas. Designen är kanske den mest kritiska faktorn som påverkar kvaliteten av systemet då den har en stor påverkan på de senare faserna, speciellt testning och underhåll. Design dokumenten kan betraktas som ritningarna till systemet.[102]

4.1.2.1 Systemdesign

Systemdesign har som mål att identifiera de moduler och komponenter som ska ingå i systemet. Moduler ska specificeras och en beskrivning över hur olika moduler samarbetar med varandra för att producera önskade resultat ska ges. Vid slutet av systemdesignen bestäms alla de större datastrukturerna, filformat och utmatningsformat. I systemdesignen

tas beslut om designen på en system nivå.[103] Enligt vattenfallsmodellen ska systemdesignen verifieras innan utvecklingsarbetet kan gå vidare till den detaljerade designfasen .

4.1.2.2 Detaljerad design

När man valt vilket slags lösning man ska använda på en systemnivå görs en detaljerad lösning som grundar sig på den aktuella utrustningen och programvaran. All systemdesign detaljeras och läggs in i den detaljerade designen. I den detaljerade designen specificeras den interna logiken för varje modul och systemutvecklare tar beslut angående hur komponenterna ska implementeras i mjukvaran.[104] Designen fokuserar på att i detalj beskriva hur systemet är organiserat på enhets- och rutinnivå. Den detaljerade designen ska enligt vattenfallsmodellen verifieras när den anses vara färdig innan systemutvecklingen går vidare till nästkommande fas.[105]

4.1.2.3 Verifiering och validering under VOV-modellens designfas

Enligt vattenfallsmodellen ska både systemdesignen och den detaljerade designen verifieras innan nästa steg i vattenfallsmodellen kan påbörjas. Om fel görs under designfasen kommer de automatiskt reflekteras i koden och det slutliga systemet. Eftersom kostnaden för att ta bort fel ökar är det viktigt att designfel upptäcks tidigt, innan de visar sig i systemet. I vattenfallsmodellen utförs endast verifiering av systemdesignen, en kontroll för att försäkra att dokumentationen fångar upp de krav som ställts i systemkravspecifikationen. Vidare utförs även endast verifiering av den detaljerade designen för att kontrollera att den stämmer överens med systemdesignen.

Användands VOV-modellen som komplement till vattenfallsmodellen utförs även validering av den designen som producerats och inte bara verifiering som beskrivits ovan. Valideringsaktiviteten försäkrar att de produkter som utarbetats är rätt för kunden och inte bara att produkterna utformas på rätt sätt. Används VOV-modellen så utförs även verifiering och validering under arbetets gång och på så sätt kan fel upptäckas ännu tidigare än om bara vattenfallsmodellen hade använts.

De verifierings- och valideringsaktiviteter som ingår i VOV-modellens designfas är genomgångar och inspektioner. Eftersom vattenfallsmodellen kräver att all systemdesign läggs in i en detaljerad design utförs förslagsvis inspektioner på de delar av designen som inbegriper viktiga eller komplexa moduler och genomgångar på de lite mindre viktiga eller enklare modulerna. Detta eftersom inspektioner tar mer resurser i anspråk än genomgångar, som kan vara mer informella.

4.1.3 Vattenfallsmodellens implementering

Vattenfallsmodellens kodning-, integration och test- och implemeneringsfas är de faser som behandlas i kombination under VOV-modellens implementeringsfas.

Under kodningsfasen implementeras designen och om designdokumentet är fullständigt går kodningsfasen i den takt den ska. Enligt vattenfallsmodellen så börjar testningsfasen så fort kodningsfasen är färdig. Testerna verifierar att mjukvaran stämmer överens med systemkravspecifikationen. En testplan skrivs för att definiera hela testningsprocessen och individuella testningsprocedurer utvecklas baserade på logisk nedbrytning av kraven. Testningsaktiviteterna dokumenteras i en testrapport. Efter att testningen är klar kan systemet levereras till kunden.[106]

Om vattenfallsmodellen kompletterats med VOV-modellen under systemutvecklingen har testbarhet byggts in i systemet på ett tidigt stadium och testning för att kontrollera att alla krav är uppfyllda ska gå lätt att genomföra. Men VOV-modellen fokuserar inte bara verifierings- och valideringsaktiviteter under implementeringsfasen till automatisk testning utan uppmärksammar även manuella genomgångar som kompletterar de automatiserade testerna för att upptäcka fel. Manuella genomgångar ska utföras innan den automatiserade testningen tar vid.

4.1.3.1 Manuella genomgångar

Vid användandet av VOV-modellen som komplement till vattenfallsmodellen så utförs först statisk analys, läsning av kod och kodinspektioner för att verifiera den programkod som producerats. Statisk analys innebär kompilering av koden för att t.ex. upptäcka eventuella fel som ska avlägsnas innan testningen inleds. Läsning av kod genomförs för att kontrollera att implementeringen inte avviker från designen. Efter att dessa båda aktiviteter genomförts kan inspektioner av koden hållas. Under inspektionen granskas koden med hjälp av en checklista för att försäkra att koden är av kvalitet och att den är konsistent med kravspecifikationen. Allt eftersom systemdesignen implementeras bör manuella genomgångar utföras för att verifiera koden. När de manuella genomgångarna har utförts tar de automatiserade testerna vid.

4.1.3.2 Automatiserade tester

Under den testfasen i vattenfallsmodellen, verifieras att systemet möter kraven som fastställts i systemkravspecifikationen. En testplan utvecklas för den övergripande verifieringssprocessen och individuella testprocedurer utvecklas genom en logisk nedbrytning av kraven. Efter detta dokumenteras testaktiviteterna i testrapporter.

4.1.3.2.1 Inledande test

Enligt VOV-modellen är testning är beroende av vilken utvecklingsmetod som används vid systemutvecklingen och därför används olika tester för olika metoder. Ett system som utvecklats enligt vattenfallsmodellen genomgår komponenttest som kan sägas bestå av två delar, enhetstest och modultest. Målet med komponenttest är att verifiera att komponenten fungerar korrekt oberoende av andra komponenter. När komponenter testas används ofta stubbning, vilket innebär att man simulerar grannkomponenter med temporära komponenter, sk stubbar. Dessa stubbar ska ersätta en komponents funktion i avseende på gränssnittet, dvs leverera och ta emot samma typer av in- och utdata som den

riktiga komponenten kommer att leverera då den är implementerad. Vid enhetstest testas varje individuell enhet för att se att den fungerar som man tänkt. Enheten som testas ska inte vara beroende av andra enheter utan testas fristående. Efter enhetstestning levereras sedan enheten för vidare modultest. En modul är en integrerad samling av enheter som på något sätt är relaterade till varandra och vid modultest testas modulen oberoende av någon annan modulapplikation.[107]

När modulen fungerar enligt intentionen fortsätter testfasen med integrationstest som även kallas delsystemtest. Detta test innebär att man på något sätt integrerar relaterade moduler till ett så kallat delsystem och testar detta. Denna testfas har också till uppgift att kontrollera gränssnittet till angränsande delsystem. Vid utvecklingen av ett system delar man ofta upp utvecklingen av systemet i delsystem som utvecklas parallellt med varandra. Man lägger stor vikt vid testning av gränssnitten mellan delsystem eftersom många av de fel som upptäcks beror på fel i gränssnitten.[108]

4.1.3.2.2 Systemtest

Systemtest av system som utvecklas enligt vattenfallsmodellen föregås av någon form av inkrementell testordning vilket innebär att börja någonstans i strukturen och sedan testa sig igenom denna steg för steg. Fel som upptäcks i någon fas kan då direkt åtgärdas vilket inte hade varit fallet om alla enheter hade slagits ihop på en gång och testat helheten. När systemtestningen är avklarad fortsätter testfasen med acceptanstest.

4.1.3.2.3 Acceptanstest

Acceptanstest är den sista fasen i testprocessen innan mjukvaran anses färdig att levereras till kund. Detta test involverar kunden som är med vid testkörningen. Testet utförs i den ”verkliga” omgivningen och valideras för att försäkra att systemet uppfyller de krav som finns i kravspecifikationen. Acceptanstestet pågår tills systemutvecklare och kund kommer överens om att det färdiga systemet fungerar enligt kravspecifikationen. Implementeringsfasen innebär att systemet levererats till kunden för användning. Vid acceptanstest upptäcks felaktigheter och missar i kravspecifikationen, program- och designfel hittas och behovet av nya funktioner identifieras. Modifiering blir en nödvändighet för att systemet ska kunna vara användbart. Genom att göra dessa ändringar så upprepar man några eller alla tidigare steg i vattenfallsmodellen.[109]

4.1.3.2.4 Regressionstest

När någon del i programkoden modifieras måste de delar av programmet som påverkats av ändringen testas om, detta kallas för regressionstest. Regressionstest av system som utvecklas enligt vattenfallsmodellen utförs t.ex. om man i integrationstestfasen upptäcker ett fel i någon komponent, korrigeras felet varefter komponenttest och modultest upprepas på hela systemet efter varje korrigering. Detta blir väldigt dyrt i praktiken och man kan istället testa en delmängd av system-komponenterna.[110]

4.1.4 Sammandrag av vattenfallsmodellen och VOV-modellen

Om vattenfallsmodellen kompletteras med VOV-modellen kan systemutvecklare och kund arbeta experimentellt med prototyper för att utforma en kravspecifikation under VOV-modellens analysfas. Krav- och systemkravspecifikationen verifieras och valideras sedan genom inspektioner kontinuerligt under analysfasen för att korrekt specificera kundens krav. Hade systemet utvecklats enbart med vattenfallsmodellen hade ingen verifiering av dokumentationen gjorts i analysfasen utan endast validering. ´

Designdokumenten som produceras under designfasen ska även valideras, och inte bara verifieras, när VOV-modellen kompletterar vattenfallsmodellen. Designen fungerar som en ritning under implementeringsfasen och får inte innehålla fel. VOV-modellen uppmärksammar att testbarhet ska byggas in i systemet på ett tidigt stadium vilket medför att testprocessen ska vara enkel att genomföra, men innan testningen påbörjas ska manuella genomgångar utföras.

4.2 Systemutveckling med objektorientering och VOV-modellen

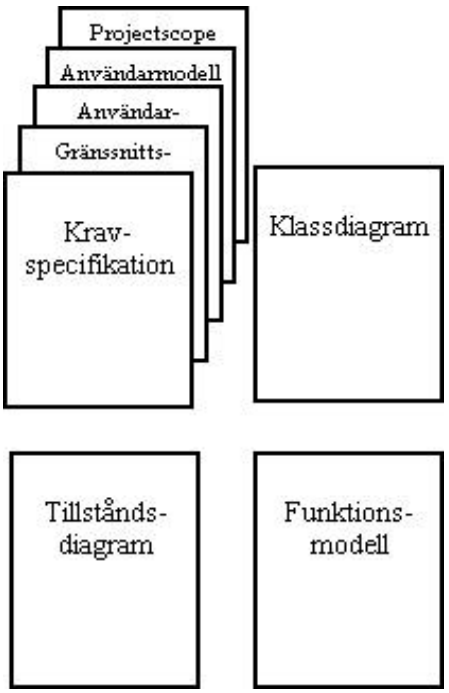
Att använda sig av en objektorienterad metod vid systemutveckling innebär att det tänkta systemet modelleras i objekt. Målet är att fånga en modell som avspeglar den verkliga världen, där objekten som modelleras finns motsvarade i ”verkligheten”. På så sätt kan de som använder modellerna som produceras under systemutvecklingen få en ökad förståelse för systemet. Det är för det flesta människor naturligt att använda sig av objekttänkandet i verkligheten vilket gör att det är lätt att ta till sig objekttänkandet. Den utvecklingsmodell som beskrivs nedan grundar sig på den beskrivning som ges i boken *”Object-Oriented Analysis: objects in plain English”*, skriven av Brown, D., 1997 där en beskrivning ges av den objektorienterade utvecklingscykeln.[111] Eftersom denna bok främst behandlar analysfasen så har vi kompletterat med uppgifter från annan generell litteratur om objektorientering.

Genom analys, design och implementering bibehålls samma representation, notation och tankesätt. Det iterativa tankesättet runt och mellan de olika faserna gör att det går lätt och mjukt att röra sig mellan de olika systemutvecklingsfaserna.[112] Genom att utveckla ett system med en objektorienterad teknik kan ett krav spåras i kravspecifikationen genom systemdesignen till koden och vice versa. Detta gör att det blir lättare att underhålla och förstå systemets uppbyggnad.[113]

4.2.1 Objektorienterad analys

Objektorienterad systemutveckling startar med en analysfas för att bestämma systemkraven, dvs vad som ska implementeras. Under denna fas skapas en formell modell av det önskade systemet vilket ger en förståelse för systemets egenskaper, krav och relation till omvärlden.[114] Syftet med analysfasen är att skapa en förståelse för vad systemet ska utföra, förbereda för förändringar, skapa ett underlag för återanvändning och skapa ett underlag för vidare design av systemet.[115]

De milstolpar som produceras under den objektorienterade analysfasen är en konceptuell modell som består av kravspecifikation, klassdiagram och tillståndsdiagram samt, vid utveckling av stora och komplexa system, funktionsmodeller.[116]



Figur 12 Kravspecifikationens beståndsdelar.

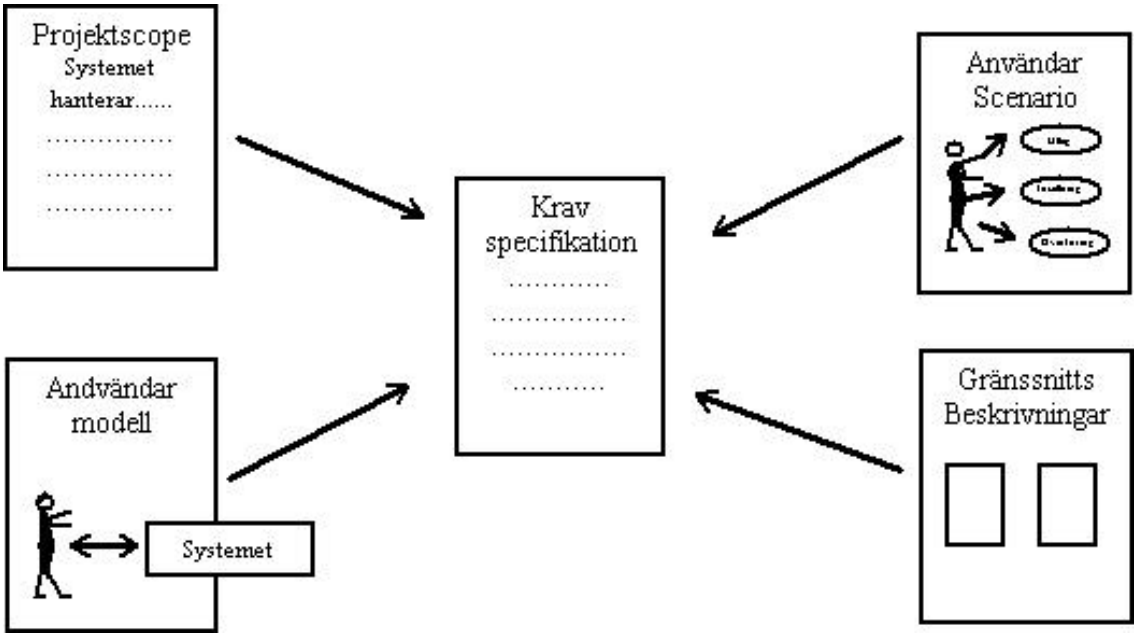
Utvecklare ska under analysfasen bekanta sig med användarnas verksamhet och fånga upp denna i modellen. Eftersom modellen ska dokumentera hur kunden vill ha systemet arbetas modellen fram i nära samarbete med kund. Modellen av systemet är relativt detaljerad, medan kraven på funktionalitet och interaktion beskrivs i form av fullständiga listor utan särskilt många detaljer.[117] De objektorienterade beskrivningarna bör kompletteras med användarinriktade framställningar och prototyper, för att användarna ska förstå vad informationssystemet innebär. När den konceptuella modellen har godkänts betraktas den som ett skriftligt avtal mellan kund och systemutvecklare.[118]

I följande stycken ges först en beskrivning av den konceptuella modellens beståndsdelar följt av en beskrivning hur dessa verifieras och valideras enligt VOV-modellen.

4.2.1.1 Kravspecifikation

Kravspecifikationens funktion är att dokumentera användarnas behov på ett sätt som gör att användare förstår dokumentet och utvecklare finner det vara användbart för att börja utveckla systemet. Kravspecifikationen är inget statiskt dokument utan kan betraktas som ett levande kontrakt mellan användare och utvecklare.[119]

I den objektorienterade metoden utformas kravspecifikationen av de fyra mindre modellerna ”project scope”, användarmodell, användarscenarion och gränssnitts-beskrivningar vilka visas i nedanstående figur:



Figur 13 Kravspecifikationens beståndsdelar i den objektorienterade metoden.

”Project scope” är det dokument som i text ger en generell beskrivning av vad systemet ska hantera. Nästa modell kallas för en användarmodell och är de ritningar som beskriver hur systemet förhåller sig till externa enheter, som t.ex. användare eller andra system. Den tredje modellen behandlar användarscenarion vilka beskriver hur systemet används och den sista modellen är en slags gränssnitts-beskrivning där utseenden på användargränssnitt och gränssnitt till angränsande system beskrivs.[120]

4.2.1.2 Klassdiagram

I analysfasen identifieras de klasser som existerar i systemet och med klassdiagrammet ges en översiktlig beskrivning av relationerna mellan dessa klasser. Ett klassdiagram beskriver den statiska strukturen av klasser i systemet. Klasserna representerar de ”saker” som hanteras i systemet.[121] Beskrivningen av en klass kan göras detaljerad genom beskrivningar av attribut och funktioner. Diagrammet kan detaljeras ytterligare genom att ange attributens typ och funktionernas parametrar, samt visa om funktionerna är offentliga eller privata.[122] Att leta klasser är ett kreativt arbete och det borde göras av experter inom problemområdet.[123]

4.2.1.3 Tillståndsdiagram

Tillståndsdiagram används som ett hjälpmedel för att gå igenom ett objekts livscykel och upptäcka dess beteende. Ett objekt kan i sin livscykel anta olika tillstånd och varje

tillstånd associeras med vissa beteenden. Objektet förändrar tillstånd genom att en utlösande händelse får objektet att byta tillstånd. Tillståndsdiagram talar om vilket tillstånd som ett objekt kan ha och hur händelserna påverkar dessa tillstånd över tiden.[124] Tillståndsdiagrammen ger viktig insyn i objekten och underlättar för utvecklaren att upptäcka funktionerna som en klass av objekt måste kunna utföra.[125] För att inte diagrammen ska bli svåra att överskåda är det bra att begränsa antalet tillståndsdiagram till att endast beskriva de viktigaste klasserna.[126]

4.2.1.4 Funktionsmodell

Funktionsmodeller används för att dokumentera beräkningar och dataöverföringar för en klass. Denna typ av modell ska endast användas om en klass har stora eller komplexa funktioner. Med hjälp av funktionsmodeller blir det lättare att beskriva en klass funktioner i en senare del av utvecklingen.[127] I flera system är det en klass funktioner som kommer att vara det som förändras mest under systemets livstid. Med tanke på detta är det viktigt att systemet byggs utifrån de objekt och klasser som finns i ”verkligheten” och som kan tänkas finnas under en tid framåt.[128]

4.2.1.5 Verifiering och validering under VOV-modellens analysfas

I VOV-modellens analysfas betraktas den konceptuella modellen som utformats i den objektorienterade analysfasen, som en milstolpe. Denna milstolpe består av de ovan beskrivna dokumenten kravspecifikation, klassdiagram, tillståndsdiagram samt funktionsmodell vilka VOV-modellen betraktar som delstolpar. Kravspecifikationen i sig består av fyra dokument (se kap 4.2.1.1) så man kan säga att det är sju dokument under analysfasen som betraktas som delstolpar av VOV-modellen. Verifiering och validering av dessa ska ske genom inspektioner, allt eftersom de skapas, under hela analysfasen. Vid användandet av VOV-modellen utförs inspektioner även under själva arbetet av dessa dokument, de behöver inte vara färdiga för att en inspektion ska utföras. En inspektion kan hållas på flera av delstolparna samtidigt då dessa dokument kompletterar varandra.

Om inspektionerna utförs som beskrivits enligt VOV-modellen finns det en deltagare som där har rollen som ”författare till kravspecifikationen” och representeras här av författaren till respektive delstolpe. Den checklista som används som hjälpmedel vid inspektionen för att upptäcka felaktigheter i kravspecifikationen kan kompletteras med exempelvis följande kontrollpunkter när systemet utvecklas objektorienterat:

- Finns det klasser som bär samma namn?
- Har alla *attribut logiska namn*?
- Finns det externa system som kan bli klasser, som ej definierats?
- Är alla aktörer/*klasser definierade*?
- Är alla *relationer* mellan klasserna definierade?
- Definieras klassernas *skyldigheter*?
- Är specifikationen *konsistent med verkligheten*?

I VOV-modellen finns möjligheten att arbeta experimentellt med prototyper, något som den objektorienterade metoden kan använda sig av. Användandet av en prototyp kan vara ett bra tillvägagångssätt för att kommunicera med kunden och utforma en kravspecifikation. Efter att prototypen har utprovats av kunden kompletteras prototypen med den dokumentation som ingår i den konceptuella modellen. Inspektioner utförs sedan för att verifiera och validera dokumentationen.

Tillsammans ska de dokument som produceras under analysfasen användas som riktlinjer för att utveckla en design.[129] De klassdiagram som produceras får inte innehålla allt för mycket information samtidigt utan måste brytas ner till en överskådlig mängd, exempelvis en A4-sida.[130] De användarscenarion och tillståndsdiagram som produceras bör kompletteras med en textuell beskrivning som exempelvis förklarar vilka situationer som hanteras och vilka krav som ställs på omgivningen etc.[131] Detta för att tydliggöra innebörden i dokumenten och underlätta för kommunikationen mellan systemutvecklare och kund angående kravspecifikationen.

4.2.2 Objektorienterad designfas

Det kan vara svårt att säga vart analysfasen slutar och designfasen börjar i den objektorienterade metoden p.g.a. otydliga gränser, eftersom samma notation och arbetssätt används.[132] Men innan designen av ett system kan påbörjas krävs en exakt och fullständig dokumentation av vad systemet ska klara av.[133] En objektorienterad design ska först och främst värderas med avseende på hur den motsvarar de specifika krav som ställts under analysfasen.[134]

Under den objektorienterade designfasen modifieras den konceptuella modellen som skapats under analysfasen. Först skapas en modell på en logisk nivå, som kallas systemdesign, vilken senare utvecklas på en mer fysisk nivå i form av fysiska designmodeller. Systemet anpassas till hård- och mjukvara, optimeringar och svarstider.[135]

Objektorienterad design strävar efter att gömma information inom ett objekt. Man vill göra objekt till självständiga enheter som medför att en förändring av ett objekt inte påverkar andra objekt. Detta medför även att de självständiga objekten lätt går att flyttas runt i designen. Objekten delar inte på variabler vilket leder till att systemet blir mer modellerbart.[136] Objektorienterad design strävar även efter att kunna återanvända klasser och det är därför eftertraktat att designa klasser som fungerar som självständiga och flexibla enheter.[137]

I designfasen kommer utvecklaren att hitta och lägga till ett antal nya objekt och klasser, som inte är objekt av entitetsklasser som identifierats under analysfasen, utan är objekt av antingen gränssnitts-, kontroll- eller abstrakta klasser. (För vidare förklaring läs Brown, 1997, kap 5 & 8). Vissa av klasserna läggs till för att förenkla programmeringen och för att återanvända kod. Ibland hittar utvecklaren redan i analysfasen dessa typer av klasser. Utvecklaren väljer då mellan att lägga till dessa klasser i ett klassdiagram direkt eller att vänta tills designfasen.[138]

Vissa typer av dokument lämpar sig bättre än andra för detaljering. Diagram kan användas för att ge en bra överblick över ett system och detta är fördelen med ett klassdiagram. Skulle designern upptäcka att en klass behövs läggas till, måste denne vara medveten om att diagrammet kan bli svårare att förstå. När dokumentet växer så finns det två alternativ för att behålla överblickbarheten, antingen visar man bara ett begränsat antal sammanhörande klasser åt gången eller att utelämna attribut från diagrammen så att bara klassnamnen är synliga. På så sätt visar dokumentet sambanden mellan ett större antal klasser.[139] Alltför mycket information i ett och samma diagram kan bli för komplext och svårbegripligt.

Milstolparna som produceras under designfasen är textdokument och fysiska modeller, vilka dels hierarkiskt visar hur klasser och subklasser förhåller sig till varandra och hur objekt samspelar med varandra.[140] Dokument som tillkommer under designfasen är textspecifikationer som ger bra beskrivningar över de detaljerade förhållandena.

4.2.2.1 Systemdesign

I objektorienterad systemutveckling används generella mönster för att designa system. Exempel på mönster kan vara skiktad arkitektur och klient-server-arkitektur. I en skiktad arkitektur ligger komponenter i olika skikt och gränssnitt uppåt och nedåt definieras. Klient-server-arkitekturen består av ett antal klienter som oberoende av varandra utnyttjar en server. Den skiktade arkitekturen är mer stram och snäv än klient-sernver-arkiltekturen som istället har ett nätverkstänkande.[141] Men det finns många mönster att välja mellan och vilket som passar bäst beror på hur systemet och infrastrukturen, t.ex. användar-gränssnitt och lagring av data, ser ut.[142]

Systemdesign lägger ut riktlinjer för hur vidare designarbete ska utföras och hanterar bl.a. beslut för hur parallella händelser ska genomföras. Objekt är i analysfasen parallella eftersom de beskriver verkligheten, men objekten måste anpassas till de praktiska begränsningar som systemet har, exempelvis användarkrav på viss utrustning eller antal processorer.[143] Objekten kommer att exekveras antingen i sekvens eller parallellt. Beslut om parallella händelser måste tas tidigt under designprocessen.[144] I systemdesignen identifieras delsystem utifrån kravspecifikationen, där kraven delats in i grupper som har något gemensamt. Man identifierar sedan delsystemens funktionalitet och gränssnitt samt den hårdvara som ska byggas i denna fas.[145] Systemdesignen ska även inkludera beskrivningar av hur man tänkt när man fattat viktiga designbeslut, vilka alternativ som funnits och orsaken till beslutet.[146]

4.2.2.2 Detaljerad design

Om det underlättar för implementeringen konstrueras en detaljerad design över ett delsystem eller en klass, tillsammans med en textuell beskrivning för varje funktion.[147] Förslagsvis ska systemutvecklare föra in varje komponent i en detaljerad design. Hur många komponenter som varje detaljerat designdokument innehåller beror på projektets storlek. Det kan vara en klass, en grupp av klasser eller ett delsystem.[148]

I den detaljerade designen specifiseras exempelvis metodnamn, returvärden och attribut. Utifrån systemdesigndokumentet förändras vissa klasser som slås ihop eller bryts ner i detalj, beroende på vilka anpassningar som måste göras. Exempel på anpassningar som den detaljerade designen utför är t.ex. förändring av relationer, skapande och borttagande av objekt, optimering samt super- och subklasser.[149]

Den detaljerade designen bör sträva efter att vara tillräckligt kortfattad och precist för att beskriva en komponents struktur, gränssnitt och funktion. Om detta uppfylls kan systemutvecklare relativt oberoende av varandra sätta igång att implementera de olika komponenterna som beskrivs i den detaljerade designen.[150]

4.2.2.3 Verifiering och validering under VOV-modellens designfas

Under den objektorienterade designfasen läggs en stor del av arbetet ned på att verifiera och validera de milstolpar som modifierats och producerats.[151] De aktiviteter som används under designfasen för detta ändamål är enligt VOV-modellen inspektioner och genomgångar av designen.

Den systemdesign som arbetas fram under den objektorienterade designfasen inbegriper dokument som ger övergripande beskrivningar på systemnivå. Dessa beskrivningar underlättar för utvecklare och kund att få grepp om systemet som utvecklas vilket förespråkas av VOV-modellen.[152] Dessa dokument är enligt VOV-modellen de milstolpar som ska verifieras och valideras under inspektioner. Frågor man kan lägga till checklistan som används vid inspektionerna är exempelvis:

- Beskriver dokumentet ett *överskådligt* antal delsystem?
- Modellerar systemet de avtalade delarna i problemområdet?
- Utnyttjas den tekniska plattformen effektivt?[153]
- Beskrivs alla subsystems ansvarstaganden?
- Beskrivs de klasser och moduler som ett subsystem består av?
- Är *kommunikationen* mellan subsystemen så *liten* som möjligt?
- Följer dokumentet utvecklingens *standard notation*? Förstår alla inblandade?
- Vilka komponenter i systemdesignen kan *återanvändas*?
- Kan systemet anpassas efter *nya krav* och behov?

De delstolpar som produceras med den objektorienterade metoden ska självklart kontrolleras med genomgångar när systemutvecklarna anser detta vara nödvändigt. Verifiering och validering ska genomföras för att kontrollera att alla delar som ska representera helheten finns med och passar in i systemdesignen samt att kontrollera att alla delar fångas upp i designmönstret.

De objekt som tillkommit under designfasen ska kontrolleras att de är flexibla. Flexibiliteten kan öka genom att införa mellanliggande objekt som medför att kopplingarna mellan objekt blir svagare vilket i sin tur leder till att systemet blir lättare att förändra. Man ska dock ta hänsyn till att mellan liggande objekt minskar systemets effektivitet då de blir ytterligare en mellanhand som ska överföra data.[154]

Detaljerad design som förändras eller skapas under designfasen behöver enligt VOV-modellen både verifieras och valideras. Detta kan ske genom genomgångar eller inspektioner av den detaljerade designen på samma sätt som systemdesignen. En kontroll ska genomföras för att försäkra att de super- och subklasser som skapats under designfasen är organiserade på ett önskvärt och funktionellt sätt. Relationerna mellan klasserna ska inspekteras för att se om de kan betraktas som inkapslade och självständiga enheter. Klassernas attribut och metoder ska kontrolleras så att de är grupperade på ett sätt som gör det enkelt att bygga ut systemet i ett framtida skede.[155]

Designen som producerats under den objektorienterade designfasen ska ge en god helhetsbild av systemet och visa att det är väl genomtänkt innan systemet realiseras under implementeringsfasen. Den detaljerade designen ska verifieras och valideras så att den stödjer implementeringen och gör att kodningen lätt kan påbörjas.[156] Ett detaljerat design dokument kommer att användas under implementeringsfasen som vägledning. Om dokumentet inte är till någon nytta för implementeringen har det inte varit av tillräckligt bra kvalitet. Det är därför önskvärt att skapa ett bra dokument som blir användbart under implementeringen.[157]

Upptäcks felaktigheter under verifierings- och valideringsaktiviteter under designfasen rättas dessa till och minskar kostnaderna jämfört med om felet hade upptäckts och avlägsnats i ett senare utvecklingsskede. Man ska vara medveten om att den detaljerade designen på en subsystemnivå kommer att behöva uppdateras och ändras ett flertal gånger under den objektorienterade implementeringsfasen.[158] Det är viktigt att verifiera och validera de ändringar som görs i den detaljerade designen genom antingen inspektioner eller genomgångar enligt VOV-modellen.

4.2.3 Objektorienterad implementeringsfas

Felaktiga designbeslut kan visa sig under implementeringsfasen. Designen strävar mot att leva upp till kravspecifikationen samt underlätta för utvecklarna att fatta implementeringsbeslut men det finns tillfällen när implementeringsverktygen inte stödjer designen. Det är då viktigt att designbesluten görs om och uppdateras i alla relevanta dokument. Utvecklarna tvingas på detta sätt att arbeta iterativt mellan design- och implementeringsfasen. Denna form av iterativt arbete kan vara ett sunt uttryck för att utvecklarna håller systemdokumenten uppdaterade och inte glömmer att ändra fattade designbeslut.[159] Systemutvecklarna som ska implementera den detaljerade designen till kod kommer vara tvungna att komplettera brister i den detaljerade designen allt eftersom de upptäcks.[160] Eftersom verifiering och validering under implementeringsfasen är en kostsam aktivitet är det viktigt att den utförs effektivt. Testplaner och testrapporter måste skrivas och utvärderas.[161] Om VOV-modellen använts för att komplettera systemutvecklingen har tesbarhet byggts in i systemet och medför att testningen underlättas.

Den objektorienterade analys- och designfasen kan följas av implementeringsfasen.[162] Under implementeringsfasen realiseras analys- och designfasens produkter i kod.

Gränsen mellan design- och analysfasen är inte skarpt avgränsad eftersom det ofta förekommer iterativt arbete mellan faserna då felaktiga designbeslut upptäcks under implementeringsfasen.

I objektorientering kan en komponent beskrivas som *”…en samling av programdelar som utgör en välavgränsad helhet och har ett väldefinierat ansvar.”* (Mattiassen, Munk-Madsen, Nielsen, och Stage, 1998, s. 367). Exempelvis betraktas en klass eller en instans av en klass som en komponent. Under implementeringsfasen fattas beslut om hur objekts identitet, tillstånd och beteende ska representeras. En grupp klasser som har en naturlig relation till varandra bildar ett kluster. I objektorienterad systemutveckling finns en stävan att göra kommunikationen mellan klustren så liten som möjligt för att på så sätt få en effektiv exekvering av systemet.[163]

4.2.3.1 Manuella genomgångar

Komponenter som verifieras och valideras i ett objektorienterat system under denna fas är vanligtvis en klass eller en instans av en klass. Enligt VOV-modellen så ska manuella verifierings- och valideringstekniker användas innan automatiserade tester tar vid. Verifiering av typer och variabler ska göras genom statisk analys, läsning av kod följt av inspektioner av kod . Manuella metoder är användbara för att hitta fel på en klassnivå.

4.2.3.2 Automatiserade tester

Automatiserad testning tar vid efter de manuella genomgångarna utförts. Enligt VOV-modellen är automatiserade tester indelade i inledande testning, följt av system- och acceptanstest.

4.2.3.2.1 Inledande test

Den inledande testningen innebär testning av komponenter. Komponentest för ett objektorienterat designat system kan variera både till utseende och omfattning.

Systemutvecklarna kan välja att först testa alla klasserna var för sig. För att testa en specifik klass måste man först skapa en omgivning bestående av de klasser som är kopplade till denna. Detta tar tid eftersom man inte bara behöver sätta sig in i hur själva klassen som skall testas fungerar, utan också hur klasserna som är kopplade till denna fungerar. Detta leder till att testplanen och dokumentationen blir krångligare och större. Det är därför oftast bättre att testa enheterna i form av s.k. kluster. Detta är en klar fördel eftersom integrationstestningen istället kan testa beteendet framför strukturen. Ett kluster definieras som en samling klasser som utför någon sorts funktion tillsammans eller på något annat sätt är logiskt relaterade. Vid testning av objektorienterade system är det systemets beteende det intressanta och inte strukturen. Vid klustertestning testas hur klasserna i klustret fungerar tillsammans och hur gränssnitten mellan dem fungerar.[164]

Klustertest inleds genom integrationstest som går till så att den klass som är minst beroende av andra klasser plockas ut som ”standardklass”. Sedan integreras denna klass med den grannklass som i sin tur är minst beroende av andra klasser. Denna integration

leder till att vi har ett litet kluster som ska testas. Fungerar detta integreras ytterligare en grannklass till klustret, varefter detta testas. Denna metod skapar själv en testomgivning allteftersom fler klasser integreras till klustret.

4.2.3.2.2 Systemtest

När alla klasser har integrerats till något kluster så sätts de olika klustren ihop stegvis och testas i ett systemtest för att kontrollera att systemet fungerar som avsett.[165] Enligt VOV-modellen ingår det i systemtestet att även validera systemet mot funktionella och icke-funktionella krav för att kontrollera att rätt system byggs.

4.2.3.2.3 Acceptanstest

När systemtesten är genomförda utförs acceptanstest som enligt VOV-modellen involverar kunden. Acceptanstest är det slutliga testet som validerar om systemet motsvarar de krav som ställts i kravspecifikationen. Detta test kan utföras en längre tid när systemet är placerat i sin slutliga omgivning.[166] Acceptanstest utförs som beskrivits i VOV-modellen.

4.2.3.2.4 Regressionstest

Reggressionstest av system som utvecklats med en objektorienterad metod utförs efter det att ett fel identifierats och korrigerats för att åter igen testa de delar som påverkats av modifieringen. Regressionstest utförs som beskrivits i VOV-modellen, men dess utförande är beroende av hur systemets design ser ut. Ett objektorienterat systems regressionstest på en högre nivå innebär att hänsyn måste tas till komplexa relationer t.ex. arv, aggregation och tillstånden hos objekten. Ett sätt att regressionstesta ett objektorienterat designat system är att dela in testmetoden i nedanstående fyra faser:[167]

1. Identifiera de komponenter som påverkats av modifieringen.
2. Bestämma sig för en strategi och en testordning som på effektivaste sätt testar de påverkade komponenterna.
3. Utföra regressionstestet genom återanvändning, modifikation och generering av tidigare testfall.
4. Implementering av regressionsplanen som ska testa de modifierade delarna av systemet.

När de klasser som ska omtestas identifierats uppstår alltså frågan i vilken ordning dessa ska testas och integreras. När ordningen för testen bestämts utförs sedan regressionstestet.

4.2.4 Sammandrag av objektorientering och VOV-modellen

Den objektorienterade metoden innebär att utveckla ett system genom att tänka i objekt som avspeglar den ”verkliga” världen. Under systemutvecklingens olika faser behålls samma representation, notation och tankesätt vilket gör det lättare att röra sig mellan de olika faserna. Vid verifierings- och valideringsaktiviteterna enligt VOV-modellen innebär detta även att systemutvecklare och kund lättare kan föra en kommunikation angående de

milstolpar som produceras, allt eftersom objekttänkandet övas upp och kunden och lär sig att känna igen notationen.

Den objektorienterade metodens kravspecifikation består i sig av flera mindre dokument som alla ska granskas genom inspektioner. Detta för att försäkra sig om att systemutvecklingen producerar kvalitativa produkter. Utifrån kravspecifikationen utvecklas sedan systemdesign där komplexa klasser ska läggas in i en detaljerad design. Verifiering och validering av dessa dokument kan enligt VOV-modellen göras antingen formellt genom inspektioner eller mer informellt genom granskningar.

Fel som upptäcks genom verifierings- och valideringsaktiviteter under de olika systemutvecklingsfaserna ska korrigeras och uppdateras i alla relevanta dokument för att hålla systemutvecklingen kontrollerbar. I den objektorienterade metoden genomförs ofta iterativt arbete där man går tillbaka till en tidigare fas. Detta kan tolkas som ett sunt tecken för att uppdateringar i alla relevanta dokument utförs. Men har uppdateringar genomförts måste ytterligare verifiering och validering genomföras, enligt VOV-modellen. Den automatiserade testningen av system som utvecklas med den objektorienterade metoden kan ha varierande utseende beroende på vilken typ av system som utvecklas men fokus ligger på att testa systembeteendet snarare än att testa strukturen.

4.3 En jämförelse vid användandet av VOV-modellen

I detta avsnitt gör vi en jämförelse mellan att använda VOV-modellen som komplement till vattenfallsmodellen, som beskrivits i kap 4.1, och till en objektorienterad metod, som beskrivits i kap 4.2. Vi undersöker de likheter respektive skillnader som dessa metoder medför vid användandet av VOV-modellen. Vi ställer alltså dessa metoder mot varandra.

4.3.1 VOV-modellens analysfas

Vid analysfasen i VOV-modellen verifieras och valideras del- och milstolpar som producerats av de båda systemutvecklingsmetoder vi beskrivit. I vattenfallsmodellen produceras två milstolpar; en kravspecifikation och en systemkravspecifikation som kan kompletteras med förtydligande dokument som betraktas som delstolpar. Den objektorienterade metoden producerar fler del- och milstolpar; en kravspecifikation, där kravspecifikationen i sig består av fyra mindre dokument, ett klassdiagram, ett tillståndsdiagram och en funktionsmodell. Alltså har systemutvecklaren här fler del- och milstolpar att verifiera och validera under VOV-modellens analysfas.

Milstolparna som produceras under vattenfallsmodellen borde vara mer komplexa och innehålla mer fakta i ett och samma dokument, än de som produceras under den objektorenterade metoden. Dock kan verifieringen och valideringen av de dokument som produceras under den objektorienterade metoden ta längre tid, eftersom det är fler dokument som måste granskas och som både kund och systemutvecklare måste sätta sig in i och förstå. Dokumenten som produceras under den objektorienterade metoden

beskriver verksamheten med hjälp av objekt. Som beskrivits tidigare i uppsatsen är det naturligt för de flesta människor att tänka i objekt, vilket medför att förståelsen för dokumenten inte borde ta så lång tid som det skulle ha gjort om det beskrivits med enheter som i vattenfallsmodellen inte nödvändigtvis är lika logiskt strukturerade som objekt.

4.3.1.1 Validering av krav genom prototyper

Prototyper lämpar sig vanligtvis bättre för den objektorienterade modellen än vid användandet av vattenfallsmodellen.[168] Men vi har funnit att genom att komplettera de båda metoderna med VOV-modellen går det i båda fallen bra att arbeta med prototyper i analysfasen. Eftersom prototyper verkar vara ett bra sätt att få fram en kravspecifikation, som både kund och systemutvecklare blir nöjda med, anser vi att båda metoderna drar fördelar av att använda prototyper för att utforma en kravspecifikaiton. Anses arbetet med prototyper inte vara lämpligt utvecklas istället enbart en kravspecifikation som sedan ska verifieras och valideras med inspektioner. Fördelar med att använda prototyper kan vara att användarna lättare kan se vad de behöver när de får sitta med vid utvecklingen och se hur prototypen utvecklas till en kravspecifikation. De får genom prototypen och systemutvecklare hjälp att utforma sina krav.

4.3.1.2 Inspektioner av krav

Båda metoderna producerar del- och milstolpar som ska verifieras och valideras genom inspektioner. Vid användandet av VOV-modellen utförs inspektioner kontinuerligt under hela analysfasen i den takt del- och milstolparna produceras.

Vid användandet av vattenfallsmodellen utan VOV-modellen som komplement så görs endast inspektioner i slutet av analys- och systemspecifikationsfaserna, när system-utvecklarna anser att specifikationerna från dessa faser är färdiga. Här görs endast validering av specifikationerna och inte verifiering, vilket medför att ingen kontrollerar att dokumentationen stämmer överens med varandra och att systemet byggs rätt. Används VOV-modellen som komplement görs inspektioner under hela analysfasen där både verifiering och validering utförs. Aktiviteterna innebär att specifikationerna blir konsistenta och att de utformas i ett nära samarbete med kunden, på så sätt uppdateras specifikationen och eventuella fel och missförstånd kan klaras ut i ett tidigt stadium. Vi anser att vattenfallsmodellen vid detta tillfälle drar nytta av att använda VOV-modellen för att verifiera och validera de del- och milstolpar som produceras.

Vid användandet av den objektorienterade metoden så finns det inte några klara gränser mellan de olika faserna i utvecklingen och det är vanligt att iterativt arbete förekommer. Om VOV-modellen kompletterar den objektorienterade metoden så måste de uppdateringar som genomförts verifieras och valideras innan utvecklingsarbetet går vidare. Detta för att försäkra sig om att uppdateringarna är gjorda på ett riktigt sätt.

Båda kravspecifikationerna som produceras i de olika metoderna ska innehålla lika betydelsefull och användbar information som kan tillfredsställa utformningen av vidare

utvecklingsarbete. Det är därför viktigt att verifiera kravspecifikationen noggrant. Det som skiljer dem åt är att den objektorienterade metoden har fler dokument att verifiera och validera, vilket medför fler inspektioner. Men ur ett annat perspektiv så innehåller inte dokumenten lika mycket information som vattenfallsmodellens dokument och lika många inspektioner på ett och samma dokument behöver troligtvis inte hållas. Det som är viktigt under VOV-modellens analysfas är inte kvantiteten av dokumenten utan det är dokumentens kvalitet som är avgörande. Använder systemutvecklare VOV-modellen som komplement skapas dokument med kvalitet och då spelar det ingen roll om det blir mycket papper. Huvudsaken är att dokumentationen är användbar och fullständig.

Vattenfallsmodellens systemkravspecifikation innehåller mycket information vilket gör att det är extra viktigt att utforma och granska den. Ytterligare ett argument för att lägga ner mycket tid åt att utforma kravspecifikationen är att vattenfallsmodellen inte tillåter iterativt arbete alltså att en senare tillbakagång ej tillåts. Inspektionernas tidsbegränsning på två timmar kan bli lite snäv och det kan vara svårt att ta till sig innehållet i dokumentet. Hinner deltagarna inte med att inspektera specifikationen på två timmar upprepas en ny inspektion, vilket medför att det blir fler inspektioner på ett och samma dokument.

Vid användandet av den objektorienterade metoden så påbörjas modellering redan under analysfasen. Modeller anser vi vara ett bra sätt att använda för att föra en kommunikation med användarna, då användarna lätt kan förstå modeller eftersom det är naturligt att tänka i objekt och då de lätt känna igen sin egen verksamhet. Används vattenfallsmodellen görs beskrivningarna med enheter som inte alltid ger lika logiska samband som vid objekttänkande, därför kan det bli svårare att få en klar förståelse för de enheter som modelleras. Detta kan medföra att det blir svårare att verifiera och validera dokumenten.

Om del- och milstolpar kontinuerligt verifieras och valideras allteftersom de produceras under VOV-modellens analysfas på det sätt som beskrivits i VOV-modellen leder detta till att båda metoderna erhåller kravspecifikationer av god kvalitet, med konsistenta och stabila krav som är testbara och realistiska. Nackdelen med vattenfallsmodellen är dock att om systemutvecklare eller kund vid en senare fas i systemutvecklingen märker att kravspecifikationen måste förändras tillåts inte detta p.g.a. att man inte får klättra tillbaka upp i vattenfallet. Det blir därför ännu viktigare i vattenfallsmodellen att verifiera och validera att kravspecifikationen är realistisk och fullständig. Vi tror att det är svårt för kunden att fullständigt kunna specificera de krav som systemet ska motsvara tidigt i utvecklingen. Kundens förståelse för vilka krav de kan ställa på systemet växer troligtvis under utvecklingstiden då de kanske upptäcker att kravspecifikationen inte är fullständig. Arbetet med att analysera vilka krav systemet ska nå upp till och skapa en lösning i designfasen borde fördelaktigt kunna ske parallellt, vilket tillåts med den objektorienterade metoden. Anledningen till detta är att när designen genomförs kommer nya frågor om problemet upp och då är det bra att kunna gå tillbaka till kravspecifikationen för att göra ändringar.

4.3.2 VOV-modellens designfas

Verifierings och valideringsaktiviteter i VOV-modellens designfas strävar efter att få en klar design med självständiga enheter och moduler så att systemet lätt kan byggas ut. Enheterna ska sträva efter att vara självständiga så att funktioner som användare efterfrågar lätt ska kunna läggas till i designen samtidigt som existerande användar-funktioner ska gå att modifieras. Den dokumentation som de båda metoderna producerar ska vara lätt att läsa och förstå av både kund och systemutvecklare. Det är därför viktigt att designdokumentationen bryts ner till att bli hanterbara delar och inte är för komplexa.

I vattenfallsmodellens system- och detaljerade designfas produceras dels en systemdesign och dels en detaljerad design vilka betraktas som milstolpar i VOV-modellens designfas. Enligt vattenfallsmodellen måste hela systemdesignen läggas in i den detaljerade designen. Systemdesignen ska verifieras och valideras innan utvecklingen kan gå vidare till den detaljerade designen.

Under den objektorienterade metodens designfas kompletteras de milstolpar som producerats under tidigare analysfas med uppgifter. Här tas även beslut angående vilket designmönster som ska användas samt hur parallella händelser ska hanteras. Dessa beslut dokumenteras dels i en systemdesign och vid behov i en detaljerad design.

När del- och milstolparna producerats under de båda systemutvecklingsmetoderna ska dessa verifieras och valideras genom granskningar och inspektioner enligt VOV-modellen. På detta sätt erhålls dokument som är konsistenta med kravspecifikationen och som är av god kvalitet.

4.3.2.1 Genomgång och inspektion av systemdesign

Vattenfallsmodellen verifierar endast systemdesignen och utför ingen validering. Används VOV-modellen som komplement till vattenfallsmodellen måste även validering av systemdesignen göras där frågan ”Bygger vi rätt system?” ställs. Vi anser att det är bra att även validering av systemdesignen utförs där systemutvecklare har en öppen kommunikation med kunden för att försäkra sig om att systemet som byggs är det kunden efterfrågar.

I den objektorienterade metoden utförs verifiering och validering av systemdesignen samt verifiering och validering av de dokument som skapats under analysen men som kompletterats under designfasen, exempelvis klassdiagram. Detta medför att det blir fler dokument som ska granskas jämfört med vattenfallsmodellen. Därför är det extra lämpligt att använda genomgångar, vilket inte kräver lika mycket planering eller mänskliga insatser som vid inspektioner. I den objektorienterade metoden finns ingen tydlig gräns mellan systemdesign och detaljerad design, jämfört med vattenfallsmodellen, och det är vanligt att iterativt arbete mellan faserna förekommer.

Skillnaden mellan de två metoderna är här att systemdesignen inte behöver läggas in i den detaljerade designen vid användandet av den objektorienterade metoden.

4.3.2.2 Genomgång och inspektion av detaljerad design

Vid användandet av vattenfallsmodellen utförs bara verifiering och ingen validering av den detaljerade designen. Används VOV-modellen kompletteras vattenfallsmodellen med validering av den detaljerade designen, på samma sätt som vi beskrivit ovan vid systemdesignen. Fel som smugit sig in i systemdesignen och som inte upptäcks där kan fortsätta in i den detaljerade designen om endast verifiering utförs. Används validering kan det finnas en större chans att någon deltagare i inspektionen eller genomgången upptäcker att dokumentationen inte stämmer överens med kravspecifikationen som gjorts i ananlysfasen. Upptäcks inte detta fel, som av misstag förts in i utvecklingen, kan det medföra att det fortsätter med in i den resterande utvecklingen.

Enligt vattenfallsmodellen måste systemdesignen läggas in i den detaljerade designen vilket däremot inte behöver göras vid användandet av den objektorienterade metoden. Detta anser vi ha både för och nackdelar. En fördel med vattenfallsmodellen är här att de beslut som tagits i den detaljerade designen dokumenteras och på så sätt är det lättare att hålla systemutvecklingen kontrollerbar. Nackdelen är att under implementeringsfasen kommer systemutvecklare troligtvis att upptäcka att vissa detaljerade designbeslut inte går att genomföra utan löser det på plats utan att uppdatera tidigare dokument då iterativt arbete ej tillåts. Genom att göra på detta sätt blir systemdokumentationen ofullständig och ger inte den rätta bilden av systemet som utvecklas. Då VOV-modellen verifierar och validerar den detaljerade designen sker detta av personer som ska ha tillräckligt med kunskap inom programmering för att avgöra om designen är realiserbar. På så sätt minskas risken att den detaljerade designen inte går att realisera.

Enligt den objektorienterade metoden behövs endast komplexa moduler och klasser föras in i en detaljerad design. Här väljer systemutvecklaren om det är nödvändigt och om den detaljerade designen ska vara formell eller informell. VOV-modellen stödjer att komplexa delar av systemdesignen förs in i en detaljerad design eftersom det blir enklare att göra framtida förändringar av systemet om koden lätt kan följas.

4.3.3 VOV-modellens implementeringsfas

De manuella genomgångarna som används för att verifiera och validera den kod som produceras under VOV-modellens implementeringsfas är statisk analys, läsning av kod och inspektion. Vid den automatiserade testningen används inledande test, systemtest, acceptanstest och regressionstest. Verifierings- och valideringsaktiviteter under implementeringsfasen i VOV-modellen anger endast att manuella genomgångar ska utföras innan de automatiserade testerna ska påbörjas. Detta för att lokalisera eventuella fel direkt och korrigera dessa innan den traditionellt dyra testningen sätts igång.

4.3.3.1 Manuella genomgångar

Vi finner att både vattenfallsmodellen och den objektorienterade metoden utför de manuella genomgångarna av implementeringen på likvärdiga sätt och finner därför inga skillnader i användandet. Vi tror att sätten att utföra de manuella genomgångarna mer beror på valet av programmeringsspråk än valet av metod. VOV-modellens manuella

genomgångar utförs med hjälp av kompilatorer vilka kan vara mer eller mindre utvecklade beroende på hur länge programmeringsspråket använts. Programmeringsspråk delas in i hög- och lågnivåspråk där högnivåspråken är lättare att läsa eftersom de är mer likt vanligt skriftspråk. Men eftersom läsning av kod och inspektioner utförs av programmerare som bör har god kunskap om programmeringsspråket så kan de troligtvis även läsa ett lågnivåspråk. Det viktiga är ändå att manuella genomgångar utförs innan automatiserade tester vid verifieringen och valideringen under implementeringsfasen

4.3.3.2 Automatiserade tester

Eftersom inledande test, systemtest, acceptanstest och regressionstest bygger på den teststrategi som guidar testningen av systemet så är det teststrategin och inte metodvalet som avgör i vilken ordning som testerna genomförs. Teststrategin bygger på den systemdesign som utvecklats och dessa skiljer sig åt beroende på om systemet utvecklats genom vattenfallsmodellen eller med den objektorienterade metoden. Enligt VOV-modellen ska den automatiserade testningen som utförs under implementeringsfasen inledas med någon form av inkrementell testordning vid den inledande testningen innan systemtest utförs.

4.3.3.2.1 Inledande test

I ett system som utvecklats med den objektorienterade metoden finns inte samma strukturella uppdelning som i ett system som utvecklats enligt vattenfallsmodellen. Komponenttest för ett system som utvecklats med vattenfallsmodellen är uppdelat i två delar, enhetstest och modultest, medan samma test för ett system utvecklat med en objektorienterad metod kan variera till både utseende och omfattning. Det intressanta i testningen av ett system som utvecklats med en objektorienterad metod är att testa hur objektens gränssnitt och hur objekten fungerar tillsammans medan ett system utvecklat enligt vattenfallsmodellen sätter korrektheten hos strukturen framför korrektheten av systembeteendet.

Ordningen på hur man integrerar de olika enheterna skiljer sig lite åt beroende på om systemet utvecklats med vattenfallsmodellen eller med den objektorienterade metoden. Detta beror på att strukturen i ett system som utvecklat med objektorientering inte har samma hierarkiska struktur som en traditionell designad applikation. Därför blir integrationsordningen mellan klasserna inte lika självklar vid det objektorienterade systemet som vid system som utvecklats enligt vattenfallsmodellen.

4.3.3.2.2 Systemtest

Systemtestning av ett system som utvecklats med en objektorienterad metod och system utvecklade med vattenfallsmodellen har huvudsakligen samma uppgift. När teststrategi väljs vid projektstarten görs detta mycket beroende på vilken struktur designen har. I ett system utvecklat enligt vattenfallsmodellen står strukturen i centrum för testningen men i ett system utvecklat med en objektorienterad metod finns inte samma logiska struktur. I ett system som utvecklats med den objektorienterade metoden göms all struktur i objekten.

4.3.3.2.3 Acceptanstest

Ett system som utvecklats med den objektorienterade metoden jämfört med ett system som utvecklats med vattenfallsmodellen skiljer sig inte nämnvärt vid acceptanstestningen. Acceptanstestningen har samma uppgift. I båda fallen validerar kund och utvecklare systemet i den ”verkliga” miljön.

4.3.3.2.4 Regressionstest

Regressionstest som utförs på en lägre nivå av ett system, som utvecklats med den objektorienterade metoden skiljer sig inte från regressionstest av ett system som utvecklats enligt vattenfallsmodellen. På de högre systemnivåerna blir dock testningen av ett system som utvecklats med en objektorienterad nivå svårare, då ett objektorienterat system innehåller fler komplexa relationer som testningen måste ta hänsyn till. Testning av ett system som utvecklas enligt vattenfallsmodellen använder sig ofta av stubbning men har systemet utvecklats med den objektorienterade metoden blir det både svårt och kostsamt att återskapa de komplexa relationerna som kan råda mellan objekt i ett objektorienterat system som t.ex. arv och aggregationer.

4.3.4 Sammandrag av jämförelsen

Under analysfasen lämpar sig de båda metoderna för att arbeta med prototyper för att utforma en kravspecifikation. Systemutvecklarna avgör om systemet som ska utvecklas lämpar sig för prototyparbetet eller om kravspecifikationen ska arbetas fram på annat sätt. Enligt vattenfallsmodellen tillåts ingen tillbakagång till tidigare faser utan resterande utvecklingsarbete bygger på de dokument som utvecklats under analysfasen. Men eftersom även den objektorienterade metodens systemutveckling bygger på arbete som gjorts i en tidigare fas är det här minst lika viktigt att all dokumentation som producerats under analysfasen kvalitetssäkras genom inspektioner. Vid inspektioner av krav-specifikationen kompletteras uppgifter till vattenfallsmodellen genom VOV-modellen med att även verifiera dokumenten, en kontroll görs för att försäkra att de är konsistenta med varandra. Eftersom den objektorienterade metodens kravspecifikation består av fler dokument, delstolpar, är det viktigt att kontrollera dessa genom inspektioner för att kontrollera att de är konsistenta med varandra. En fördel med den objektorienterade metoden är att systemet modelleras i ”verkliga” objekt, vilket kan ge kunden en ökad förståelse av systemet.

Både vattenfallsmodellens och den objektorienterade modellens systemdesign verifieras och valideras enligt VOV-modellen. Man strävar efter att få en klar design med självständiga enheter och moduler som gör att systemet i framtiden lätt ska gå att bygga ut eller modifiera. Vattenfallsmodellen kompletteras genom VOV-modellen med att även validera både systemdesignen och den detaljerade deisgnen, en kontroll görs med kunden för att försäkra att rätt system byggs. Vattenfallsmodellen för in all systemdesign i en detaljerad design som verifieras och valideras enligt VOV-modellen. Detta behöver inte göras i den objektoreinterade metoden där endast komplexa delar läggs in i en detaljerad design.

Under implementeringsfasen utförs manuella genomgångar och automatiserade tester för att kvalitetssäkra systemet. VOV-modellens manuella genomgångar skiljer sig inte åt beroende på om systemet utvecklas enligt vattenfallsmodellen eller med den objektorienterade metoden utan de utförs på liknande sätt. De automatiserade testerna utförs däremot olika då systemdesignen har olika utseende beroende på om systemet utvecklats genom vattenfallsmodellen eller med den objektorienterade metoden. Båda metoderna utför inledande test, systemtest, acceptanstest och regressionstest för att verifiera och validera systemet. Det som skiljer metoderna åt i denna fas är bl. a att testordningen av enheter inte är lika självklar, vid den objektorienterade metoden som vid vattenfallmodellen, och att i regressionstestet så är det svårare att testa system på högre systemnivåer om de utvecklats med den objektorienterade metoden.

Utifrån denna jämförelse kan vi konstatera att VOV-modellen medför att båda metoderna kompletteras med aktiviteter som gör att systemen får bättre kvalitet än om de hade gjorts endast med en enskild systemutvecklingsmetod. Verifierings- och valideringsaktiviteterna ges större uppmärksamhet genom att använda VOV-modellen bl.a. eftersom inspektionerna ska schemaläggas. Vad vi får tänka på är att användandet av VOV-modellen medför att kostnaden under tidiga systemutvecklingsfaser ökar, men att det troligtvis lönar sig i längden att använda den som komplement, som en självständig aktivitet. De likheter och skillnader vi utläst av de två olika metoderna väger ut varandra och att de båda har fördelar och nackdelar vid användandet av verifiering- och valideringsaktiviteter.

5. SLUTSATSER

Syftet med denna uppsats har varit att förklara hur viktigt det är att använda sig av verifierings- och valideringsaktiviteter under systemutvecklingens alla faser samt att undersöka vilka likheter och skillnader som förekommer när en objektorienterad metod och vattenfallsmodellen kompletteras med en självständig verifierings- och valideringsmodell.

Arbetet med denna uppsats har lett oss till slutsatsen att det är viktigt att använda verifierings- och valideringsaktiviteter under hela systemutvecklingen om målet är att få system som är användbara för kunden. Vi har kommit fram till att det finns vissa likheter och skillnader vid användandet av vattenfallsmodellen och den objektorienterade modellen när VOV-modellen använts som komplement, men de skillnader som finns är inte så avgörande för att kunna konstatera att en av metoderna är bättre än den andra. Vad vi kan konstatera är att VOV-modellen är ett bra komplement för att uppmärksamma vikten med att utföra ett grundligt utvecklingsarbete.

För att kunna dra några slutsatser angående hur användbar VOV-modellen är i verkligheten, för att utveckla system som motsvarar kundens krav, måste detta undersökas i praktiken. Det är svårt att utifrån litteraturstudier avgöra dess användbarhet.

Under arbetets gång har vi uppmärksammat några infallsvinklar som kan vara intressanta att utreda närmare. Det skulle vara intressant att undersöka hur VOV-modellen fungerar i praktiken:

- Leder VOV-modellen till att de system som skapas erhåller bra kvalitet utan att leda till att utvecklingen blir för dyr?

- Är företag intresse av att lägga ner resurser på att utföra kontrollaktiviteter i tidiga systemutvecklingsfaser, eller prioriterar de hellre testning mot slutet av utvecklingsarbetet?

6. REFERENSLISTA

Böcker

Alvesson, M., Sköldberg, K. (1994) *Tolkning och reflektion*. Lund : Studentlitteratur.

Andersen, E. S. (1994). *Systemutveckling - principer, metoder och tekniker*. Lund: Studentlitteratur.

Avdic, A. (1996). *Riktlinjer för rapportering*. Högskolan i Örebro:Institutionen för Ekonomi, Statistik och ADB.

Behrooz, N., Svensson, M., & Warrén, P. (1999). *En orientering i objekt*. Lund: Studentlitteratur.

Blomqvist, R., Haeger, T. (1996). *Kvalitetsutveckling- kunddriven verksamhetsutveckling i teori och praktik*. Göteborg: IHM FörlagAB.

Boehm, B.W. (1981). *Software engineering economics.* Englewood Cliffs. N.J.: Prentice – Hall.

Bowen, J. P., & Hinchey, M.G. (1999). *High-Integrity System Specification and Design*. London: Springer.

Brown, D. (1997). *An introduction to Object-Oriented Analysis : objects in plain English*. New York: Wiley.

Cronholm, S. (1998). *Metodverktyg och användbarhet –en studie av datorstödd metodbaserad systemutveckling*. Linköping: Univ., Department of Computer and Information Science.

Ejvegård, R. (1996). *Vetenskaplig metod*. Lund : Studentlitteratur.

Eriksson, H. E., & Penker, M. (1998). *UML toolkit*. New York :Wiley.

Fagerström, J. (1995). *Objektorienterad analys och design – en andra generationens metod*. Lund: Studentlitteratur.

Fransson, M. & Quist, J., (1995) *Lönsamhet och kvalitet*. Karlstad: SIS KvalitetsForum.

Jacobson, I. (1992). *Object-Oriented Software Engineering – a Use Case Driven Approach*. Wokingham : ACM Press : Addison-Wesley.

Jalote, P. (1997). *An Itegrated Approach to Software Engineering*. New-York: Springer.

Jönsson, K. (1995) *Kvalitetsboken*. Malmö: Glerups Förlag.

Mathiassen, L., Munk-Madsen, A., Nielsen, P. A., & Stage, J. (1998). *Objektorienterad analys och design*. Lund: Studentlitteratur.

McConnell, S. (1998). *Software Project Survival Guide*. Washington: Microsoft Press.

Nilsson, U. (1994). *Utveckla kvalitet.* Falkenberg: SIS Kvalitetsforum.

Perry, W. (1995). *Effecive Methods for Software Testing*. New York : Wiley.

Rakitin, S. R. (1997). *Software Verification and Validation – a Practioner´s Guide*. Boston: Artech House.

Sandholm, L. (1999). *Kvalitetsstyrning med totalkvalitet- verksamhetsutveckling med fokus på totalkvalitet*. Lund: Studentlitteratur.

Sommerville, I. (1995). *Software Engineering*(5:e rev.uppl.). Wokingham: Addison-Wesley.

Wiedersheim-Paul, F. och Eriksson, L.T. (1991) *Att utreda, forska och rapportera* Malmö:Liber-Hermods

Tidskrifter

Arthur, J., Gröner, M., Hayhurst, K., och Holloway, C. (1999) Evaluating the Effectiveness of Independent Verification and validation. *IEEE CS Press*, vol.32, 79-83

Bohlin, S. (1996).Adedatas ekonomisystem klarade inte kundens krav. *Computer Sweden*, 59

Fagan, M.E. (1999). Design and code inspections to reduce errors in program development. *IBM System Journal*, vol.38, 258-288

Kung D.C, Gao J., och Hsia P. (1995). Class firewall, test order, and regression testing of object-oriented programs. *Journal of Object-Oriented Programming*, 6, 51-64,

Lidfeldt, T. (1998). Du kan inte testa fram kvalitet *Datateknik* , 98-07

Lottson, A. (1995). Testning gör utvecklaren slarvig. *Computer Sweden*, 41

Lottson, A. (1996). Hälften programmerar utan metod: Mycket objekt men lite orientering, *Computer Sweden*, 71

Parrow, J. (1998). Programmeraren som Schaman. *Forskning och framsteg* , 2

Penker, M. (1995). Skippa alla långa metoddiskussioner: Så väljer du en objektorienterad metod. *Computer Sweden*, 42

Svensson, D. (1999) Användarnas erfarenheter måste byggas in i nya gränssnitt, *Datateknik*, 99-04

Sigfried, S. (1995) Livsfarligt slopa kravspecifikationen. *Computer Sweden*, 46

Sjögren, N. (1997). Akademiska sjukhuset förlorar miljoner på utebliven patientavgift. *Computer Sweden*, 80

Fotnoter

[1] Lottson, A. (1996), nr 71
[2] Parrow, J. (1998), nr 2
[3] Bohlin, S. (1996), nr 59
[4] Sjögren, N. (1997), nr 80
[5] Andersen, E. S. (1994), s 467
[6] Lidfeldt, T. (1998), nr 7
[7] Sommerville, I. (1995), s 460
[8] Lottson, A. (1995), nr 41
[9] Parrow, J. (1998), nr 2
[10] Lottson, A. (1995), nr 41
[11] Lottson, A. (1996), nr 71
[12] Wiedersheim-Paul, F. och Eriksson, L.T. (1991), s 150
[13] Alvesson, M. och Sköldberg, K. (1994), s 48
[14] Wiedersheim-Paul, F. och Eriksson, L.T. (1991), s 150 ff
[15] Ibid
[16] Ibid, s 62
[17] Ibid, s 27
[18] Ejvegård, R. (1996), s 15 ff
[19] Lottson, A. (1996), nr 71
[20] Wiedersheim-Paul, F. och Eriksson, L.T. (1991), s 82
[21] Avdic, A. (1996), s 65
[22] Wiedersheim-Paul, F. och Eriksson, L.T. (1991), s 82
[23] Ejvegård, R. (1996), s 16
[24] Ibid, s 67 ff
[25] Andersen, E. S. (1994), s 41
[26] Penker, M. (1995), nr 42
[27] Andersen, E. S. (1994), s 9
[28] Cronholm, S. (1998), s 59
[29] Andersen, E. S. (1994), s 48
[30] Ibid, s 12
[31] Ibid, s 49
[32] Mathiassen, L., Munk-Madsen, A., Nielsen, P. A., och Stage, J. (1998), s 368
[33] Cronholm, S. (1998), s 93
[34] Sommerville, I. (1995), s 7
[35] Behrooz, N., Svensson, M., & Warrén, P. (1999), s 8
[36] Sommerville, I. (1995), s 8
[37] Andersen, E. S. (1994), s 335
[38] Jalote, P. (1997), s 39
[39] Sommerville, I. (1995), s 5
[40] Ibid, s 6
[41] Andersen, E. S. (1994), s 467
[42] Behrooz, N., Svensson, M., & Warrén, P. (1999), s 11
[43] Ibid
[44] Svensson, D. (1999), nr 99-04
[45] Lidfeldt, T., (1998), nr 98-07
[46] Sommerville, I. (1995), s 6
[47] Jalote, P. (1997), s 33
[48] McConnell, S. (1998), s 29
[49] Ibid, s 30
[50] Ibid
[51] Ibid, s 79
[52] Ibid, s 126

[53] Jalote, P. (1997), s 78
[54] Ibid
[55] McConnell, S. (1998), s 219
[56] Sommerville, I. (1995), s 8
[57] Sandholm, L. (1999), s 27
[58] Ibid, s 163
[59] McConnell, S. (1998), s 126
[60] Mathiassen, L., Munk-Madsen, A., Nielsen, P. A., och Stage, J. (1998), s 138
[61] Ibid, s 207
[62] Ibid
[63] Ibid, s 139
[64] Sommerville, I. (1995), s 612
[65] Nilsson, U. (1994), s 45
[66] Sigfried, S. (1995), nr 46
[67] Fagan, M.E. (1999), nr 38
[68] Jalote, P. (1997), s 183
[69] Andersen, E. S. (1994), s 468
[70] Ibid, s 405
[71] Ibid, s 406
[72] Ibid, s 347
[73] Fagan, M.E. (1999), nr 38
[74] Jalote, P. (1997), s 134
[75] Ibid, s 250
[76] Sommerville, I. (1995), s 219
[77] McConnell, S. (1998), s 187
[78] Jalote, P. (1997), s 345
[79] Ibid, s 369 f
[80] Ibid
[81] Ibid
[82] Jacobson, I. (1992), s 309
[83] Jalote, P. (1997), s 403
[84] Ibid, s 468
[85] Sommerville, I. (1995), s 452 ff
[86] Ibid, s 449
[87] Ibid
[88] Jacobson, I. (1992), s 310
[89] Sommerville, I. (1995), s 448
[90] Jalote, P. (1997), s 38 ff
[91] Ibid
[92] Ibid
[93] Ibid
[94] Ibid
[95] Ibid
[96] Bowen J.P., och Hinchey M.G. (1999), s 3
[97] Ibid
[98] Rakitin, S. R. (1997), s 15
[99] Jalote, P. (1997), s 36
[100] Ibid, s 39
[101] Bowen J.P. och Hinchey M.G. (1999), s 4
[102] Jalote, P. (1997), s 38
[103] Ibid, s 15
[104] Ibid, s 16
[105] Rakitin, S. R. (1997), s 15
[106] Ibid
[107] Sommerville, I. (1996), s 449

[108] Ibid
[109] Jacobson, I. (1992)
[110] Jalote, P. (1997), s 39
[111] Brown, D. (1997)
[112] Fagerström, J. (1995), s 68
[113] Ibid, s 227
[114] Ibid, s 43
[115] Ibid
[116] Brown, D. (1997), s 163
[117] Mathiassen, L., Munk-Madsen, A., Nielsen, P. A., och Stage, J. (1998), s 208
[118] Ibid, s 354
[119] Bowen J.P., och Hinchey M.G. (1999), s 164
[120] Brown, D. (1997), s 164
[121] Eriksson, H. E., och Penker, M. (1998), s 17
[122] Mattiassen, L., Munk-Madsen, A., Nielsen, P. A., och Stage, J. (1998), s 393
[123] Eriksson, H. E., och Penker, M. (1998), s 67
[124] Ibid, s 122
[125] Brown, D. (1997), s 166
[126] Fagerström, J. (1995), s 162
[127] Brown, D. (1997), s 166
[128] Fagerström, J. (1995), s 17
[129] Brown, D. (1997), s 166
[130] Fagerström, J. (1995), s 223 f
[131] Ibid
[132] Ibid, s 68
[133] Ibid, s 45
[134] Mathiassen, L., Munk-Madsen, A., Nielsen, P. A., och Stage, J. (1998), s 205
[135] Brown, D. (1997), s 440
[136] Sommerville, I. (1996), s 248
[137] Fagerström, J. (1995), s 219
[138] Brown, D. (1997), s 167
[139] Mattiassen, L., Munk-Madsen, A., Nielsen, P. A., och Stage, J. (1998), s 283
[140] Sommerville, I. (1996), s 272
[141] Mattiassen, L., Munk-Madsen, A., Nielsen, P. A., och Stage, J. (1998), s 230
[142] Fagerström, J. (1995), s 72
[143] Ibid, s 69
[144] Sommerville, I. (1996), s 248
[145] Ibid, s 32
[146] McConnell, S. (1998), s 145
[147] Fagerström, J. (1995), s 84
[148] McConnell, S. (1998), s 196
[149] Fagerström, J. (1995), s 84
[150] Ibid, s 355
[151] Mattiassen, L., Munk-Madsen, A., Nielsen, P. A., och Stage, J. (1998), s 331
[152] McConnell, S. (1998), s 145
[153] Mathiassen, L., Munk-Madsen, A., Nielsen, P. A., och Stage, J. (1998), s 206
[154] Fagerström, J. (1995), s 84
[155] Ibid, s 93
[156] McConnell, S. (1998), s 176
[157] Mattiassen, L., Munk-Madsen, A., Nielsen, P. A., och Stage, J. (1998), s 355
[158] McConnell, S. (1998), s 195
[159] Mattiassen, L., Munk-Madsen, A., Nielsen, P. A., och Stage, J. (1998), s 367 f
[160] McConnell, S. (1998), s 201
[161] Jacobson, I. (1992), s 313
[162] Mattiassen, L., Munk-Madsen, A., Nielsen, P. A., och Stage, J. (1998), s 367

[163] Ibid, s 219 ff
[164] Sommerville, I. (1996), s 450
[165] Jalote, P. (1997), s 446
[166] Jacobson, I. (1992), s 312
[167] Kung D.C, Gao J., och Hsia P. (1995), nr 6
[168] Andersen, E. S. (1994), s 409