EKONOMIHÖGSKOLAN
Lunds universitet
Institutionen för Informatik

Mjukvarudesign i mindre projekt

Avvikelser från allmän teori

Kandidatuppsats, 15 högskolepoäng, SYSK01 i Informatik

Framlagd: 21/06/2010

Författare: Raoul Pennerup
Carl Ripa

Handledare: Anders Svensson

Examinatorer: Bo Andersson
Paul Pierce

Titel: *Mjukvarudesign i mindre projekt: Avvikelser från allmän teori*

Författare: Raoul Pennerup, Carl Ripa

Utgivare: Institutionen för Informatik

Handledare: Anders Svensson

Examinatorer: Bo Andersson, Paul Pierce

Publiceringsår: 2010

Uppsatstyp: Kandidatuppsats

Språk: Svenska

Nyckelord: *mjukvaruutveckling, objektorienterad design, små projekt*

Abstrakt

Den objektorienterade litteraturen gör sällan skillnad på om design utvecklas för små eller stora projekt. Därför ville vi undersöka huruvida små projekt medvetet gör avvikelser från vad som i övergripande delen av teorin anses vara god design. Vi ville dels undersöka vilka dessa avvikelser var, och dels motiven bakom dessa designaktiviteter.

Studien i denna uppsats omfattar empiri som är insamlad från ett litet respektive ett stort projekt. Empirin har genererats med hjälp av semistrukturerade intervjuer samt undersökningar av respektive projekts designlösningar. För att strukturera upp intervjuer och undersökningar har en grundlig litteraturgranskning genomförts, där vi försöker komma fram till vad god design egentligen är.

Resultatet av studien visade på skillnader mellan respektive projekt – det lilla projektet avvek medvetet från objektorienterade designaktiviteter i syfte att spara tid under implementationen. Det stora projektet avvek mer sällan från objektorienterad teori, då de ansåg sig spara tid under drift av slutprodukten. I vår analys kommer vi fram till att flest avvikelser sker från *förståelse* och *modularitet***,** vilka vi tolkar som de mest tidskrävande designegenskaperna att uppnå. I vår slutsats föreslår vi att behovet av god objektorienterad design grundar sig i såväl projektkomplexitet som behovet att återvända till implementation och drift.

Tack till

Våra informanter på IT Visioner och Journal Digital AB, som ställt upp på våra intervjuer samt tillhandahållit dokument för oss att granska. Vi vill även tacka vår handledare Anders Svensson, samt Sofia Lindén och Marie Warngard, som löpande under uppsatsens utformning tillhandahållit feedback på vår uppsats. Slutligen vill vi tacka Elisabeth Salonen Ripa för språkgranskning av vår uppsats.

/Carl & Raoul

INNEHÅLL

1 INLEDNING

1.1 Bakgrund

Design är ett väl använt begrepp som återfinns i en mängd olika områden. Att skapa en solid och väl tillämpad design över en artefakt är inte alltid utan motgångar och svårigheter. Detta kan beskrivas genom att citera Antoine de Saint-Exupery i Robinson (2004, pp.235):

> *"A designer knows he has arrived at perfection not when there is no longer anything to add, but when there is no longer anything to take away"*

Vidare tar Robinson (2004) upp att en framgångsrik design kan uppkomma från ett antal möjliga situationer. Men för att nå denna framgångsrika design skriver Robinson (2004) att en övergripande struktur på designprocessen i de flesta fall ligger till grund. Den tidigaste formen av denna struktur är av en linjär typ, men enligt Robinson (2004) är denna struktur något som de flesta författare finner otillräcklig. Istället argumenterar han för att metodologin är mer i form av en cykel som itererar genom varje fas av processen.

Begreppet objektorienterad design är något som har uppstått ur den evolution av mjukvaruutveckling som har pågått de senaste 30 åren. Med denna objektorienterade utveckling följde, enligt Britton & Doake (2005), en ombyggnad av olika områden inom design för att inte komplikationer och svårigheter skulle uppkomma i strukturen av en design. Begrepp som *cohesion* och *encapsulation* blev kännetecken för objektorientering och så kallade mönsterlösningar (*design patterns*) blev ett allt vanligare förekommande fenomen.

Som Hsueh et al. (2007) nämner, används mönsterlösningar för att behandla triviala men även mer vanligt förekommande problem som uppstår inom objektorienterad systemutveckling. Dessa mönsterlösningar är inte bara positiva i sammanhanget utan för även, som Hsueh et al. (2007) nämner, med sig problem då de främst är baserade på att lösa strukturella problem och inte att svara på själva användarens krav.

Förutom de nackdelar med mönsterlösningar som Hsueh et al. (2007) nämner, är den negativa kritiken mot objektorienterad utveckling relativt sparsmakad i vetenskapliga diskussioner. Det förekommer dock ett fåtal artiklar som tar upp denna klyfta mellan det objektorienterade tankesättet och det verkliga utförandet ute i projekt. Men dessa artiklar saknar dessvärre ofta en empirisk grund att utgå ifrån.

1.2 Problem och forskningsfråga

Utifrån vår hypotes, att små projekt i små företag medvetet avviker från de strikta regler som den objektorienterade synen på god design förespråkar, har vi identifierat ett problem; sällan i teorin för god mjukvarudesign görs skillnader mellan små och stora projekt. Regler för exempelvis dokumentation, mönsterlösningar och andra grundpelare inom objektorientering gäller för det stora såväl som det lilla projektet. Tidigare erfarenheter har visat på att det mycket väl kan existera medvetna avvikelser. Ett problem är dock att det mesta av litteraturen inte behandlar detta fenomen. Motiven bakom avvikelserna verkar ha att göra med den tid det

tar att producera en fullt objektorienterad design, något som mindre konsultföretag kanske inte alltid har råd med. Ur detta problemområde kommer därför vår forskningsfråga:

> *Avviker små projekt medvetet från den allmänna uppfattningen av god mjukvarudesign, och vad är motivet bakom dessa designval?*

1.3 Syfte

Med denna uppsats vill vi påvisa att utveckling i mindre projekt tar genvägar som strider mot objektorienterade principer när det gäller design av mjukvara. Vår empiriska undersökning och en efterföljande analys kommer att synliggöra vissa avvikelser från den allmänna teori som gäller för objektorienterad god design, baserat på våra studieobjekt.

Vi kommer med denna uppsats framhäva både fördelar och nackdelar med dessa eventuella avvikelser, och så långt som resultatet tillåter, analysera motiven bakom dessa designval.

1.4 Avgränsningar

Uppsatsen kommer att fokusera på de designaktiviteter, snarare än hela utvecklingsmodeller, som förekommer inom objektorienterad utveckling i mindre projekt. Det är framförallt två bidragande faktorer som ligger till grund för vår avgränsning:

- Bredden på det teoretiska området design.
- Generaliserbarheten av vårt problemområde.

Undersökningen i denna uppsats kommer inte att fokusera på effekterna av eventuella avvikelser, utan fokus kommer att ligga på avvikelserna och motiven bakom dem.

2 Mjukvarudesign & objektorientering

I detta kapitel presenteras ett antal definitioner på små projekt. Uppsatsen går sedan vidare in på det allmänna området design. Syftet med delkapitel 2.3-2.9 är att skapa en uppfattning vad den allmänna synen på god mjukvarudesign är. Följande delkapitel bidrar till att skapa denna uppfattning:

2.3: Svårigheter med design – Om svårigheter kan undvikas, så är det en bättre design.

2.4: Konsekvenser av dålig design – Ju större konsekvenser, ju viktigare är designaktiviteter för att motverka dem. Vilka är dessa aktiviteter?

2.5: Att undvika dålig design – Vilka aktiviteter som leder till god design.

2.6: Objektorienterad design – Vilka designkvaliteter den objektorienterade designen förbättrade.

2.7: Vad är bra objektorienterad design? – Konkreta designkvaliteter hos en god objektorienterad design.

2.8: Mönsterlösningar – Lösningar som anses vara bra designade. Vilka designkvaliteter som gör mönsterlösningar bra.

2.1 Små projekt

Det är svårt att exakt definiera vilka kriterier som ligger till grund för vad ett litet projekt är. Anledningen är att olika författare definierar små projekt på olika sätt. Med hjälp av litteratur kan vi således endast approximera vad ett litet projekt är.

Andersson & Rexfeldt (1999) skriver att det är svårt att definiera vad ett litet projekt är, men kommer med följande förslag på kriterier för ett litet projekt:

- I ett litet projekt har projektledaren flera roller. Som exempel kan han/hon få fungera som projektledare och linjechef samtidig.
- Projektledaren har hand om mer än ett projekt åt gången.
- Resurserna är begränsade.
- Kommunikationsvägarna mellan projektmedarbetare är kortare.

Andersson & Rexfeldt (1999) fann att den generella uppfattningen om definitionen av ett litet projekt är att tidsperioden är mindre än ett år, budgeten mindre än en miljon kronor, arbetade timmar 200-1500 och antal deltagare en till fyra. Resultatet visade även att projektets komplexitet påverkar om det är ett litet eller stort projekt. Mätbarhet av detta framgick inte från resultatet.

Melissa et al. (2000) skriver att små projekt ofta har ett större antal externa relationer per projektmedarbetare. Som exempel tar de upp att små projekt ofta har en närmre koppling till sina kunder. Vidare argumenterar de för att små projekt ofta måste uppnå kvalitetsmål som är lika strikta som de i större projekt, men med färre projektmedarbetare. De skriver även att små

projekt ofta innehåller projektmedarbetare som endast arbetar i projektet på deltid, vilket kräver mer koordination och interaktion.

Det finns flera miljörelaterade faktorer som bestämmer om ett projekt ska klassificeras som litet eller stort. Organisationens storlek påverkar om ett projekt ska ses som litet eller ej. Ett projekt i en stor organisation kan få intern hjälp med såväl resurser som kunskap, medan ett projekt i en liten organisation inte kan få samma interna hjälp. (Melissa et al., 2000)

Melissa et al. (2000) påpekar även att ett projekts storlek kan klassificeras efter hur sofistikerad kunskap som krävs för att genomföra projektet. Ju mer komplex problemdomänen är, desto större behov finns det att strukturera projektets aktiviteter. Vidare är små projekt ofta involverade med mindre kod, vilket betyder att de lättare kan uppnå uppsatta kvalitetsmål (Melissa et al., 2000).

Andersson & Rexfeldt (1999) påstod, som tidigare nämnts, att kommunikationsvägarna mellan projektmedarbetare är kortare. Melissa et al. (2000) utvecklar detta resonemang genom att påstå att kommunikationen i små projekt ofta är informella och att själva kommunikationen kan nå ut snabbare än i stora projekt på grund av dess relativt ytliga ledningsstruktur. Detta leder till att små projekt reagerar snabbare på förändringar än stora projekt (Melissa et al., 2000).

Ett exempel på en konkretare karakterisering av små projekt är:

> *”Eligible project, either emergency or permanent work, with a damage dollar value of less than $52.000.”* (U.S. Department of Homeland Security, 2009)

2.2 Design

Designmetodologi är ett relativt nytt begrepp och uppkom inte förrän så sent som på 1960-talet (Zhu, 2005). Däremot har konceptet design funnits med oss i tusentals år från grottmålningar till mer nutida ritningar av artefakter så som mjukvara. Några nutida beskrivningar av design är:

> *”formgivning, dvs. gestaltning av hantverksmässigt eller industriellt framställda produkter och miljöer. I dagligt tal avser ordet vanligen en produkts form eller utseende”* (Nationalencyklopedin, 2010)

> *“to prepare the preliminary sketch or the plans for (a work to be executed), esp. to plan the form and structure of: to design a new bridge.”* (Dictionary.com, 2010)

> *“the art or process of making a drawing of something to show how you will make it or what it will look like”* (Longman Dictionary, 2010)

Gemensamt för ovanstående karakteriseringar är att design beskrivs som en förberedande aktivitet för att planera skapandet av en designartefakt. Lawson (1980) är noga med att poängtera att design inte endast är artefakten av ett designarbete, utan även processen att skapa denna.

Dasgupta (1989) sammanfattar ovanstående definitioner genom att karakterisera design som skapandet av en lösningsplan eller en modell av ett pågående arbete innan lösningen materialiserats. Vidare beskriver Dasgupta (1989) design som ett hjälpmedel till implementationen av en artefakt, plan eller process.

Zhu (2005) skriver att generering av idéer är en vanligt citerad egenskap hos design. Zhu (2005) refererar Reswicks karakterisering; design är en kreativ aktivitet som involverar skapandet av något nytt, något som inte tidigare existerat.

Litteratur kring designmetodologi uppkom, som tidigare nämnts, först kring 1960-talet, men har under åren utformats till en egen vetenskaplig disciplin. Behovet av designmetodologier har uppmärksammats allt mer på grund av designaktiviteternas problematiska natur. Asimow beskriver designaktiviteters problematiska natur som beslutsfattande under hög osäkerhet med höga straff för felaktiga åtgärder. (Zhu, 2005)

Zhu (2005) kopplar designproblematiken till mjukvarudesign och skriver att uppkomsten av designmetodologier inom mjukvarudesign formats från praktiska erfarenheter så som att tidigare försummade designaktiviteter endast lett till problem under senare steg i utvecklingsprocessen.

2.3 Svårigheter med design

Design är en av de svåraste aktiviteterna inom mjukvaruutveckling (Zhu, 2005). Mycket av litteraturen pekar på samma anledningar till varför just mjukvaruutveckling är så svår. Brooks (1987) argumenterar för att följande egenskaper kring mjukvara gör design speciellt svårt. Även Budgen (1995) och Zhu (2005) lyfter fram liknande egenskaper som problematiska.

En mjukvaras komplexitet skapar flera svårigheter. Att entiteter kan ha flera tillstånd, att tillståndens storlek kan variera och att de kan vara interrelaterade, gör att det är svårt att förstå, beskriva och testa mjukvara (Brooks, 1987). Zhu (2005) lägger till att mjukvara har ett stort antal tillstånd och att mjukvaruentiteter är komplexa på grund av deras antal element och hur dessa element interagerar med varandra.

Komplexiteten i problemet som ska lösas skapar även svårigheter i kommunikationen mellan användare och utvecklare. Detta bidrar sedan till fel i kravspecifikationen, vilket i sin tur leder till utveckling av fel mjukvara. Dessutom försvåras kommunikationen mellan medarbetare, vilket leder till produktfel och ökade kostnader. (Zhu, 2005)

Zhu (2005) skriver vidare att komplexiteten inte endast skapar tekniska problem, utan även ledningsrelaterade problem – att få en överblick över mjukvaran blir svårt, samt att lärandeprocessen och förståelsen försvårar cirkuleringen av personal.

Sammanfattningsvis skriver Budgen (1995) att komplexiteten inom mjukvarudesign ofta skapar så kallade *wicked problems* – på grund av de många tillstånden och funktionerna i ett system leder ett problems lösning ofta till nya problem, som måste tas hand om.

Ett annat stort problem kring designen av mjukvara ligger i att mjukvara måste anpassas för flera externa objekt (Brooks, 1987). Zhu (2005) anger exempelvis att mjukvara måste anpassas för hårdvaran den ska agera i, redan existerande mjukvara och inte minst för användarna och andra mänskliga system.

Zhu (2005) tillägger att mjukvara lider av ett ständigt påliggande krav på föränderlighet. Exempel på föränderlighet är krav på uppdateringar och utvecklingen av ny teknik.

En unik egenskap med mjukvarudesign är att dess designartefakter saknar tydlig visibilitet. Till skillnad från andra artefakter än mjukvara, kan mjukvarudesign inte representeras som fysiska modeller i form av ritningar, kartongmodeller, etcetera (Zhu, 2005). Författaren fortsätter sitt resonemang med att skriva att de metoder som används för att beskriva mjukvara saknar en tydlig visuell länk och kan därmed inte förmedla förhållanden mellan designen och systemet på ett enkelt sätt. Framförallt leder detta till försvårad kommunikation mellan personer (Zhu, 2005).

2.4 Konsekvenser av dålig design

Flera författare inom mjukvarudesignsdisciplinen har identifierat liknande konsekvenser av dålig design. Här tas de till synes vanligaste och mest accepterade konsekvenserna upp.

Budgen (1995) skriver att den ultimata egenskapen för alla system är huruvida det passar in i sin kontext. Kan inte produkten lösa det problem den är skapad för så är det en dålig design. Budgen skriver vidare att endast när detta mål är uppnått kan andra faktorer övervägas. Även Zhu (2005) har uppmärksammat denna funktionalitet och förklarar att felaktighet är konsekvensen av misstolkning av uppsatta funktionella krav.

Ett annan brett accepterad konsekvens av dålig design är bristen på förståelse. Olika konsekvenser inom brist på förståelse tas upp i litteraturen, varav en är motsägelse. Motsägelse är när en design inte fungerar, exempelvis på grund av att två designpåståenden strider mot varandra. Zhu (2005) ger exempel på detta i form av att om olika påståenden strider mot varandra om vad ett dataobjekt egentligen är så kommer designen inte att fungera.

Om en design kan tolkas på olika sätt eller om den är för otydlig, kan fel i implementation av designen ske på grund av felaktiga tolkningar av den (Zhu, 2005). Avslutningsvis visar Soni et al. (2009) i sin undersökning att förståelsen av ett system lätt påverkas av strukturen i en objektorienterad design.

Zhu (2005) hävdar, att om en design inte uppnår uppsatta kvalitetskrav, kan låg anpassningsmöjlighet och ineffektivitet uppstå. Zhu (2005) beskriver brist på anpassningsförmåga som svårigheten att förändra en mjukvarudesign, och ineffektivitet som barriären som försvårar implementationen av designen. Dessa resonemang sammanfattar författaren inom begreppet underlägsenhet.

En av de kanske mest dokumenterade och undersökta konsekvenserna av dålig design är svårigheter som uppstår i samband med drift. Dålig designstruktur, så som antalet metoder, aggregeringar och brist på dokumentation, påverkar drift negativt (Soni et al., 2009). Britton & Doake (2005) har uppmärksammat att brist på inkapsling leder till försvårad drift. Chatzigeorgiou et al. (2008) har identifierat att låg abstraktion, bland annat påverkar mönsterlösningars drift negativt.

Avslutningsvis anser Agrawal & Khan (2009) att säkerhetsåtgärder bör implementeras redan under designfasen. De hävdar att antivirus och brandväggar inte är tillräckliga åtgärder för att skapa en säker produkt och refererar till flera undersökningar som pekar på att

säkerhetsåtgärder tidigt i utvecklingen kan reducera extra säkerhetsåtgärder senare i utvecklingsarbetet.

2.5 Att undvika dålig design

2.5.1 The axioms of design

Witt et al. enligt Zhu (2005) tar upp fyra påståenden som bör ligga till grund för en god design. Zhu (2005) skriver att de döptes till *axioms* (självklara) för att de var just självklara.

- *The axiom of separation of concerns* – Ett komplext problem löses bäst genom att det delas upp i mindre självständiga problem.

- *The axiom of comprehension* – Upptäckten om att en människa endast kan komma ihåg fem till nio slumpmässiga tal åt gången tas också upp. Witt et al. enligt Zhu (2005) påstår att människan endast kan manipulera ungefär sju saker åt gången.

- *The axiom of translation* – En korrekt design ska inte bli mindre korrekt för att den flyttas mellan två liknande kontexter. Designen kommer fortsatt att uppfylla uppsatta specifikationer så länge den nya omgivningen erbjuder liknande tjänster. Som exempel tas följande upp: Ett program som fungerar i en korrekt testmiljö ska fungera på liknande sätt i produktionsmiljön.

- *The axiom of transformation* – En bra design ska fortsätta att vara korrekt även om en komponent byts ut mot en annan, förutsatt att den nya komponenten uppfyller de krav som den förra komponenten uppfyllde.

2.5.2 Designprinciper

Witt et al. enligt Zhu (2005), utgick från ovanstående självklarheter och utvecklade ett antal designprinciper som var och en ska lösa olika designmål. Några av dem presenteras här.

- *The principle of modular designs* – Modularitet kan uppnås genom att dela upp stora samlingar av komponenter till mindre enheter. Uppdelningen ska leda till att enheterna har en låg intern koppling och en hög sammanhållning. På så sätt kan modifieringar av enheter ske utan att de påverkar varandra.

- *The principle of portable designs* – Genom att abstrahera den verkliga kontexten kan portabilitet uppnås.

- *The principle of malleable* (formbara) *designs* – Formbarhet kan uppnås genom att designen modellerar slutanvändarens omgivning.

Zhu (2005) vill göra ett tillägg till ovanstående principer. Som nämnts tidigare är osynlighet ett stort problem inom mjukvarudesign och bör därför ligga till grund för följande princip:

- *The principle of visualization* – Synlighet kan uppnås genom att representera designer med hjälp av visuella notationer så som diagram, bilder och figurer – som förklarar ett systems egenskaper och beteenden, samt relationerna mellan komponenter.

2.6 Objektorienterad design

2.6.1 Uppkomsten av objektorientering

För att undersöka varför den objektorienterade designen uppkom måste vi först titta tillbaka till varför det strukturella tillvägagångssättet bedömdes otillräckligt.

Britton & Doake (2005) skriver att under många år, innan uppkomsten av objektorienterad design, utvecklades mjukvara med hjälp av det strukturella tillvägagångssättet *functional decomposition*. Med detta menar Britton & Doake (2005) att varje komponent som utvecklades hade en direkt länk till en viss aktivitet den skulle genomföra. Som exempel tas ett motorcykelinhyrningssystem upp. I detta skulle förmodligen en process skapas för varje aktivitet som kan uppstå; hyra cykel, lämna tillbaka cykel, och så vidare. Med *functional decomposition* fanns det en klar separation mellan process och data. Varje process utför någon aktivitet och data är fritt tillgänglig och överförbar. Detta är ett av de större problemen med det strukturella tillvägagångssättet; data kan hämtas och manipuleras hur som helst (Britton & Doake, 2005).

Britton & Doake (2005) tar även upp ett antal problem med detta tillvägagångssätt. De är enligt följande:

- *Drift* – Strukturerad mjukvara som använder *functional decomposition* medför stora driftproblem. Eftersom all data är tillgänglig överallt kan modifiering av viss kod medföra problem någon annanstans i koden. Britton & Doake (2005) refererar till detta problem som *the ripple effect*. Utvecklare uppmärksammade detta och kom fram till att data måste skyddas (inkapslas) från obehörigt tillträde.

- *Begränsad modularitet* – Enligt Britton & Doake (2005) bör ett systems komponenter vara isolerade från varandra med ett tydligt syfte och de bör vara oberoende från varandra. Dessa egenskaper besatt inte *functional decomposition*.

- *Testning* – Eftersom komponenter var beroende av varandra var det svårt att utföra isolerade tester på specifika komponenter.

- *Återanvändning* – Eftersom komponenter var beroende av varandra var det svårt att återanvända kod. Detta eftersom beroendena inte fungerade i komponentens nya miljö.

- *Funktionsberoende* – Eftersom systemen som byggdes var baserade på funktionalitet snarare än data, var det svårt att skapa system som inte ständigt behövde modifieras. Detta eftersom verklighetens processer förändras oftare än dess data. Som exempel tas ett sjukhus upp. Sjukhus förändrar ibland sina processer så som patientinläggning, men data förblir densamma – patienter är fortfarande patienter och sängar är fortfarande sängar. Detta ledde till att utvecklare förstod att system baserade på data vore stabilare.

2.6.2 Fördelar med objektorienterad design

Med ovanstående historiska problem i åtanke formulerades följande fördelar med det nyare objektorienterade tillvägagångssättet (Britton & Doake, 2005). De nya objekten skulle vara:

- Självständiga från andra objekt.
- Sammanhängande med ett enda väl definierat syfte.
- Lätta att förstå.
- Lätta att anpassa för att möta nya eller förändrade krav.
- Baserade på data.

De skulle även ha:

- Skyddad (inkapslad) data.
- Ett väl definierat publikt interface.

2.6.3 Kritik mot objektorientering

Mansfield (2005) diskuterade kring varför objektorienteringen blivit så utbredd och därmed varför tillgänglig litteratur oftast är för objektorientering. Han tar även upp ett antal objektorienterade påståenden som han anser vara felaktiga. Artikeln, om än intressant, baseras mycket på påståenden, men på grund av brist på kritisk litteratur väljer vi att ta med den.

Produktivitet är inte längre den största drivkraften vid systemutveckling och det finns inte några belägg för att objektorientering skulle vara bättre än procedurbaserad design. Mansfield följer upp detta med att påpeka att ingen debatt kring objektorienteringens nytta existerar och att detta beror på ”akademiska teorister” som ser till att varje nyutexaminerad student är formad i den objektorienterade andan. (Mansfield, 2005)

Vidare argumenterar Mansfield (2005) för att objektorienterade termer så som inkapsling, arv och polymorfism är bra i teorin, men svåra att applicera i verkligheten. Dessutom går det att uppnå mycket med procedurbaserad design, som även går att uppnå genom objektorientering.

Skov & Stage (2002) undersökte huruvida objektorientering förhindrar kreativt skapande, främst när det gäller *story telling*. Den objektorienterade metoden drogs med ett antal svagheter. Det var svårt med dynamiken i situationsbaserade objekt, då simuleringar av verkliga situationer baserade på användarnas aktiviteter utfördes. Dessutom tillförde användningsfall ingen kunskap om situationer i verkligheten. (Skov & Stage, 2002)

2.7 Vad är bra objektorienterad design?

Som nämnts tidigare bör en design huvudsakligen värderas emot huruvida den uppfyller sitt syfte (Budgen, 1995). Litteraturen tar dock upp ett antal egenskaper som kan användas för att bedöma om delar av en design är bra eller dålig. Budgen förklarar vidare att designkvalitet är en subjektiv term, som betyder olika för olika roller. Kunden vill kanske ha bra prestanda och snabb leverans, programmeraren kanske vill ha tydliga riktlinjer och försäljaren vill eventuellt ha så många produktegenskaper som möjligt.

Ilighoven (2005) anser att mjukvaruutveckling alltid bör ses som ett steg för att uppnå ett högre syfte, vilket ofta är att stödja kundrelaterade affärsprocesser och tjänster i en organisation eller ett företag.

Binkley & Schach (1996) hävdar att termen designkvalitet måste specificeras. Binkley & Schach (1996) menar även att designkvalitet främst mäter huruvida designen uppfyller uppsatta specifikationer, men också hur designen påverkar implementation och drift.

Sevarani, et al. (2010) argumenterar för att strukturen i en objektorienterad design har en tydlig koppling till dess designkvalitet. Designen måste enligt Sevarani, et al. (2010) inneha egenskaper så som analyserbarhet, föränderlighet, stabilitet och testbarhet, vilka är viktiga för driften av systemet. Vidare hävdar de att komplexiteten i systemet (polymorfism, inkapsling och beroenden) försvårar driften av systemet.

Inkapsling

Objekt bör skydda sin data mot felaktig hantering av denna i syfte att motverka korruption. Om andra objekt ska hämta eller manipulera ett objekts data så ska detta ske med hjälp av operationer. (Britton & Doake, 2005)

Budgen (1995) är inne på samma spår och hävdar att det högsta målet med inkapsling är att begränsa och kontrollera åtkomst till information inom systemets struktur så att dess detaljerade form hålls skyddad. Detta leder till förenkling när det gäller att förändra datastrukturer utan sidoeffekter (Budgen, 1995).

Beroende

Ett av de vanligaste förekommande måtten på designkvalitet inom mjukvarudesign, enligt den teori vi funnit (delkapitel 2.6.2 och 2.7), är hur beroende olika objekt är av varandra.

Beroende (*coupling*) beskriver hur moduler interagerar med varandra. Ett bra beroende har proceduranrop med klart specificerade parametrar och väl definierade ingångs- och utgångspunkter. Dålig koppling är till exempel *GOTO*-kommandot – att i koden gå vidare till annan kod genom angivelsen av radnummer (Budgen, 1995).

Om två moduler använder varandras tjänster så sägs de vara beroende (Britton & Doake, 2005). De argumenterar för att beroenden måste kontrolleras eftersom moduler som är beroende av varandra teoretiskt sätt kan manipulera varandra.

Binkley & Scach (1996) hävdar vidare att ett vanligt sätt att mäta kvalitet på en klass är att räkna hur många klasser som den kan interagera med. De argumenterar för att detta mått är något godtyckligt. Ska beroende användas som mått på kvalitet så bör typen av kopplingar det rör sig om identifieras - om parametrar skickas *by value* eller *by reference* (som nytt värde eller som referens till det gamla värdet) (Binkley & Schach, 1996).

Vidare har beroende använts som kvalitetsmått i flera år och de drar slutsatsen att beroende fortfarande, år 1996, kan användas som basen till kraftfulla mått för objektorienterad design (Binkley & Scach, 1996). Soni et al. (2009) påvisar detta i sin undersökning; direkta kopplingar mellan klasser har en stark negativ påverkan på återanvändbarheten för en

komponent. Deras undersökning visade att mer än 84 % av undersökningspersonerna höll med om detta påstående.

Arv

Arv är en användbar teknik endast om den används på rätt sätt. Arv mellan klasser ska inte förekomma endast för att låta den ena klassen få tillgång till den andra klassens funktionalitet. Klasser ska endast ärva från andra klasser om det finns ett ”är en”- eller ”är en slags”-relation mellan dem. (Britton & Doake, 2005)

Inte olikt ovanstående argument skriver Britton & Doake (2005) att i en arvshierarki ska subklasser alltid vara utbytbara mot superklassen.

Binkley & Schach (1996) påpekar att det finns flera mått baserade på arvshierarki. Djupet i en arvshierarki, antalet klasser som ärver från en viss klass, samt bredden på en arvshierarki är exempel på mått som används. Dock förklarar Binkley & Schach (1996), som i fallet med beroende, att undersökningsobjekten måste studeras mer i detalj. Som exempel nämner Binkley & Schach (1996) att dessa mått varken tar hänsyn till multipla arv trots att multipla arv har stor negativ effekt på både utveckling och drift.

Även Soni et al. (2009) tar upp arv som ett kvalitetsmått. Precis som Binkley & Schach (1996), skriver Soni et al. att arvskvalitet kan mätas på olika sätt. Soni et al. (2009) har identifierat antalet arvshierarkier som främst ett funktionalitetsmått, medan djup, aggregering och antalet subklasser är mer lämpliga för mätning av effektivitet.

Agrawal & Khan (2009) tar ett mer säkerhetsinriktat perspektiv på arv. De förklarar att om en klass ärver sårbar funktionalitet eller data från en superklass, kommer även subklassen att vara sårbar. De fortsätter med att påstå att arv är ansvarig för spridning av sårbarheter från klass till klass på grund av dess transitativa natur.

Sammanhållning

Britton & Doake (2005) hävdar att en klass har hög sammanhållning om den endast behandlar en sak och om dess attribut och operationer relaterar till samma ämne. Vidare beskriver de att ett bra objekt har hög sammanhållning – att det endast berör ett enda uppgiftsområde och därmed blir lämpliga för återanvändning.

Även Budgen (1995) påstår att sammanhållning har att göra med i vilken utsträckning som komponenter i en modul behandlar samma mål och uppgifter. Vidare påstår Budgen att även om sammanhållning är en subjektiv term, kan den användas som kriterium för god design.

Avslutningsvis visar Soni et al. (2009) i deras undersökning att sammanhållning påverkar en design på flera olika sätt. Främst påverkar sammanhållningen förståelsen av designen, men sammanhållningen påverkar även återanvändbarheten av komponenter samt funktionaliteten i systemet. Soni et al. (2009) skriver att hög polymorfism har samma effekter.

Domän och funktionalitet

Klasser ska överensstämma med verkliga entiteter i problemdomänen, vilket underlättar läsbarhet av kod (Britton & Doake, 2005). Robinson (2005) hävdar att läsbarheten av kod kan förstärkas genom dokumentation inuti koden.

Vidare anser Britton & Doake (2005) att klasser ska ha såväl data som funktionalitet. Utvecklaren ska hålla ögonen öppna efter klasser som endast är en enda stor funktion och inte har några attribut (Britton & Doake, 2005). Vidare är en klass utan funktionalitet förmodligen en dåligt designad klass. De skriver även att funktionaliteten i system delas mellan klasser och bör således utföra mer än att endast visa värden på attribut. Dock ska man, enligt Soni et al. (2009), vara aktsam för att specialisera funktionalitet genom att skapa för många klasser. Enligt Soni et al. (2009) har för många klasser en negativ påverkan på funktionalitet.

2.8 Mönsterlösningar

Hsueh et al. (2007) beskriver mönsterlösningar (*design patterns*) som populära sätt att kapsla in kunskap om objektorienterad design. De skriver även att mönsterlösningarna beskriver lösningar till återkommande designproblem som uppstår vid utveckling av mjukvarusystem. Med återkommande problem menas generella men inte triviala problem.

Cote et al. (2007) beskriver mönsterlösningar som sätt att förstå ett givet designproblem eller som ett verktyg för att strukturera problemets lösning. På så sätt kan konstruktionen av *throw-away models* undvikas, vilka är modeller som kastas bort och byggs upp på nytt (Cote et al., 2007).

Mönsterlösningar beskriver problem som ständigt återkommer i vår omgivning samt deras lösning. Vidare kan en mönsterlösning användas om och om igen och dessutom beskriver den varför denna bör användas till ett visst givet problem. (Voran, 2009)

Voran (2009) sammanfattar även det som ovan nämnda författare sagt genom påståendet att mönsterlösningar beskriver instanser av bra design.

2.8.1 Fördelar

Att identifiera och katalogisera mönsterlösningar kan öka produktiviteten eftersom designern inte behöver lösa liknande problem flera gånger. På liknande sätt kan organisationer bibehålla kunskap genom att spara och katalogisera mönsterlösningar, vilket är särskilt nyttigt i stora organisationer. (Voran, 2009)

Chatzigeorgiou et al. (2008) beskriver en undersökning utförd av Prechelt, som visade på att användningen av mönsterlösningar underlättade driftarbetet både gällande tidsåtgång och antal uppkomna fel. Undersökningen replikerades av samma Prechelt, vilket gav liknande resultat. Ett liknande test utfördes ännu en gång av Prechelt med flera, vilket även den visade på att användningen av mönsterlösningar förkortade tidsåtgången vid driftaktiviteter. Ghezzi et al. i Chatzigeorgiou et al. (2008) skriver att det primära målet med att applicera mönsterlösningar är att underlätta drift.

Borchers (2001) och Erickson (2000) i Chatzigeorgiou et al. (2008) påpekar att mönsterlösningar stödjer och förbättrar kommunikationen mellan projektmedlemmar från olika discipliner. Chatzigeorgiou et al. (2008) motiverar detta med att mönsterlösningar skapar ett gemensamt språk som används vid förklaring och diskussion av designlösningar.

Undersökningen i Chatzigeorgiou et al. (2008) visade att antalet kopplingar mellan objekt samt antalet funktioner som kommunicerar med andra objekt minskade. (Hsueh et al, 2007) anser även att den sekundära effekten av användandet av mönsterlösningar (den primära effekten är att lösa ett visst problem) är att minska antalet kopplingar mellan objekt. Hsueh et al. (2008) identifierar även inkapsling som en fördel med att använda mönsterlösningar.

2.8.2 Nackdelar

Chatzigeorgiou et al. (2008) hävdar att mönsterlösningar minskar kodkomplexitet i designlösningar. Dock visar deras resultat raka motsatsen; undersökningspersonerna ansåg mönsterlösnignarna vara svåra att förstå. Dessutom visade sig användandet av mönsterlösnignar öka flera storleksrelaterade mått så som antal rader kod, antal klasser, antal objekt och antalet attribut.

2.9 Sammanställning

För att skapa ett underlag för vår undersökning har vi valt att sortera och kategorisera flera av de designkvaliteter som går att identifiera från teorin. Denna sammanställning kommer användas dels som underlag för vad som kommer undersökas under intervju och dokumentundersökning, och dels som underlag för presentation av resultat samt diskussion.

Vår frågeställning grundar sig i hypotesen att små projekt gör vissa medvetna designavvikelser från vad som allmänt ses som god design och att dessa skulle kunna motiveras med vinst i tid. Teorin säger, som vi visat under teorikapitlet, att god design underlättar implementation, men framförallt drift. Därför har vi valt att skilja på egenskaper som underlättar implementation och drift. Vi har även valt att kategorisera designkvaliteter under testning, då testning av erfarenhet sker under såväl implementation som drift.

Teorin har pekat på ett antal designkvaliteter som kan uppnås genom god design. Vidare beskriver teorin dessa designkvaliteter som resultatet av olika designaktiviteter. Dessa designaktiviteter ämnar vi undersöka. Sammanställningen (Tabell 2.4) beskriver vilka variabler som påverkar vilka designkvaliteter samt vilka designkvaliteter som hör ihop med olika utvecklingsfaser.

2.9.1 Implementation

Funktionalitet – Beskriver, som nämnts tidigare (delkapitel 2.7), huruvida de funktionella kraven uppfylls eller ej. Detta innefattar dels att all funktionalitet är implementerad, och dels att rätt entiteter gör rätt saker. Ur teorin kan slutsatsen dras att antalet klasser har en koppling till en designs funktionalitet.

Förståelse – Vi har tidigare visat (delkapitel 2.4 och 2.7) att förståelsen beskriver hur lätt en design kan tolkas och sedan implementeras. Ett stort antal beroenden kan försvåra förståelsen.

Koddokumentation tillsammans med talande klasser ökar utvecklarens förståelse, särskilt om ny personal tillsätts i ett redan existerande projekt. Att entiteter har en verklighetsförankring ökar även det förståelsen.

Storlek – Storleken av en implementation är direkt kopplad till antalet rader kod (delkapitel 2.8). Antal och typ av arv påverkar storlek. Vidare påverkar antalet kopplingar och användandet av koddokumentation implementationens storlek. Genom att använda talande klasser, med talande metoder och attribut kan, som tidigare nämnts, förståelsen av en design ökas. Dock ökar även storleken på implementationen.

Tabell 2.1: Implementationsrelaterade designkvaliteter

Utvecklingsfas	Designkvaliteter	Variabler
Implementation	Funktionalitet	Klass: Antal
		Klass: Data & Funktionalitet
		Sammanhållning
	Förståelse	Beroende
		Koddokumentation
		Sammanhållning
		Talande klasser
	Storlek	Arv: Bredd
		Arv: Multipla
		Beroende
		Klass: Antal
		Koddokumentation

2.9.2 Testning

Modularitet – Beskriver utbytbarheten och därmed isoleringen av olika modulers egenskaper och funktionalitet, vilket leder till att testning kan ske på isolerade moduler i systemet. I teorin har vi visat att modularitet främst påverkas av antalet beroenden, men att även multipla arv och utbytbarheten av arv påverkar modularitet. Vidare leder hög grad av inkapsling till en högre grad av modularitet. Ett högt antal klasser påverkar modulariteten negativt, medan en hög sammanhållning leder till högre modularitet. (delkapitel 2.5.2 och 2.6.1)

Tabell 2.2: Testningsrelaterade designkvaliteter

Utvecklingsfas	Designkvaliteter	Variabler
Testning	Modularitet	Arv: Multipla
		Arv: Utbytbarhet
		Beroende
		Inkapsling
		Klass: Antal
		Sammanhållning

2.9.3 Drift

Förståelse – En lättförståelig design och därmed lättförståelig implementation underlättar drift. För mer detaljer om förståelse se beskrivning i delkapitel 2.9.1.

Modularitet – En hög modularitet underlättar drift. Detta främst då delar av ett system behöver bytas ut eller modifieras, men modulariteten underlättar även sammansättningen av nya moduler. För mer detaljer om modularitet se beskrivning i delkapitel 2.9.2.

Säkerhet – Vi har tidigare visat (delkapitel 2.4) att djupet på en arvshierarki kan påverka ett systems säkerhet negativt. Om subklasser ärver sårbar data eller funktionalitet från en huvudklass så ska även dessa subklasser betraktas som sårbara.

Tabell 2.3: Driftrelaterade designkvaliteter

Utvecklingsfas	Designkvaliteter	Variabler
Drift	Förståelse	Beroende
		Koddokumentation
		Sammanhållning
		Talande klasser
	Modularitet	Arv: Multipla
		Arv: Utbytbarhet
		Beroende
		Inkapsling
		Klass: Antal
		Sammanhållning
	Säkerhet	Arv: Djup

Med ovan identifierade implemenations-, testnings- och driftrelaterade designkvaliteter har vi skapat följande teoretiska ramverk för vad som allmänt uppfattas som god mjukvarudesign.

Tabell 2.4: Sammanställning av designkvaliteter

Utvecklingsfas	Designkvaliteter	Variabler
Implementation	Funktionalitet	Klass: Antal
		Klass: Data & Funktionalitet
		Sammanhållning
	Förståelse	Beroende
		Koddokumentation
		Sammanhållning
		Talande klasser
	Storlek	Arv: Bredd
		Arv: Multipla
		Beroende
		Klass: Antal
		Koddokumentation
Testning	Modularitet	Arv: Multipla
		Arv: Utbytbarhet
		Beroende
		Inkapsling
		Klass: Antal
		Sammanhållning
Drift	Förståelse	Beroende
		Koddokumentation
		Sammanhållning
		Talande klasser
	Modularitet	Arv: Multipla
		Arv: Utbytbarhet
		Beroende
		Inkapsling
		Klass: Antal
		Sammanhållning
	Säkerhet	Arv: Djup

3 METOD

3.1 Metodbeskrivning

Med vår forskningsfråga och vårt syfte i åtanke hade vi bestämt oss för en viss arbetsgång. Vi valde att arbeta utifrån Jacobsens (2002) undersökningsprocess, vilken vi anser är en logisk och metodisk process för att utifrån en frågeställning kunna besvara en viss fråga. Dessutom grundar sig Jacobsens (2002) undersökningsprocess i att faser sker stegvis och att de påverkar varandra, något vi kan relatera till på följande sätt:

Vi formulerade en frågeställning vi ville undersöka. Utifrån frågeställningen har lämpligt underlag i form av teori samlats in för att utforma en framtida undersökningsmall. Val av undersökningsmetod har gjorts, vilken vi beskriver mer detaljerat senare i detta kapitel.

Val av undersökningsmetod grundade sig i den frågeställning vi ville besvara. Vi valde att utföra ett antal semistrukturerade intervjuer samt dokumentundersökningar, vilka vi beskriver närmare senare i kapitlet.

3.2 Val av undersökningsobjekt

Jacobsen (2002) skriver att en *intensiv uppläggning* är lämplig när många variabler ska undersökas. Med en *intensiv uppläggning* menar Jacobsen (2002) en undersökning med många variabler och få enheter att undersöka. Han motiverar denna uppläggning med att resurser sällan finns tillgängliga för att undersöka många variabler hos många enheter.

På liknande sätt har vi resonerat när vi valt undersökningsobjekt. Vi hade inte resurserna att undersöka våra variabler på flera observationsobjekt. Vi har istället valt att fokusera på en djupgående, snarare än en bred, undersökning. Vi motiverar detta val främst med att vi ville påvisa företeelser snarare än att generalisera upptäckter, något vi inte skulle klara av med hjälp av en extensiv (Jacobsen, 2002) uppläggning. Vi har således valt att göra en undersökning av följande observationsobjekt:

IT Visioner är ett systerbolag till IT-gården AB. IT Visioner står för mjukvaruutvecklingen i IT-Gården AB. På grund av att IT Visioner är ett nystartat och litet företag (två anställda), utför de endast små projekt inom mjukvaruutveckling och lämpar sig således som undersökningsobjekt för vår undersökning.

Journal Digital Sverige AB tillhandhåller ett system bäst beskrivet som:

> *"Ett verksamhetsstöd som integrerar all den dokumentation, planering och utvärdering som krävs av socialstyrelsen för att journalföra och mäta effekter av psykosocialt behandlingsarbete"*. (Journal Digital AB)

Utvecklingsteamet är liksom IT Visioner ganska litet, men projektet har pågått i flera år och har blivit stort och kunskapskraven på utvecklarna har blivit höga. Med kriterierna för vad som är ett litet projekt (delkapitel 2.1) ser vi detta projekt som medelstort till stort. Journal

Digital AB kommer i vår undersökning användas som jämförelse till IT Visioner. Fokus kommer fortfarande att ligga på det lilla projektets medvetna avvikelser.

De båda företagen med tillhörande projekt uppfyllde våra krav på undersökningsobjekt; ett projekt skulle vara litet, ett skulle vara stort och mjukvaruutveckling skulle praktiseras. Den externa giltigheten stärks om undersökningen är överförbar mellan olika observationsobjekt (Jacobsen, 2002). Dock används våra undersökningsobjekt annorlunda. Det lilla projektet visar på avvikelser från allmän teori, medan det större ger kompletterande syn på god design.

3.3 Val av informanter

Inför urvalet till vår intervju resonerade vi kring vilken person i ett litet företag som vore mest lämplig för vår undersökning. På grund av att vi valt undersökningsobjekt innan vi valt informanter, i kombination med att våra undersökningsobjekt har få involverade projektmedarbetare, lämnades inte särskilt stort utrymme för önskemål av informant. Nedanstående persona beskriver vår förhoppning om informant:

En erfaren utvecklare eller en projektledare med utvecklarerfarenhet. Personen får gärna ha utvecklat i både stora och små projekt. Kunskap och åsikter om nyutexaminerade utvecklares styrkor/svagheter är önskvärt.

Vår första uppgiftslämnare är chefen på IT Visioner. Han är personligen ansvarig för alla pågående projekt och även väldigt aktiv som utvecklare. Han har erfarenhet av såväl små som stora projekt i små som stora företag.

Vår andra uppgiftslämnare är utvecklare på Journal Digital AB. Han är nyutexaminerad systemvetare, men har det övergripande ansvaret för företagets mjukvaruutveckling.

3.4 Semistrukturerade intervjuer

Intervjuerna skedde ansikte mot ansikte. Intervjuer utförda ansikte mot ansikte är mer givande och uppgiftslämnaren har lättare att diskutera känsliga frågor (Jacobsen, 2002). Vidare skriver Jacobsen (2002) att intervjuer genomförda ansikte mot ansikte tillåter uppgiftssökaren att mer aktivt tolka uppgiftslämnarens fysiska reaktioner. Dock anser han att telefonintervjuer är mindre resurskrävande.

Vi utförde intervjuerna där uppgiftslämnarna ville att de skulle genomföras. Samtliga intervjuer med IT Visioner skedde på deras kontor i Landskrona, medan intervjuerna med Journal Digital AB skedde hemma hos en av uppsatsförfattarna. Vidare spelades samtliga samtal in och transkriberades, vilket uppgiftslämnarna efter förfrågan inte hade något emot. En öppen intervju kommer resultera i väldigt mycket data (Jacobsen, 2002). Vi anser att det skulle varit väldigt svårt att till fullo lyckas anteckna allt som sades utan inspelning.

Vi definierade ett antal diskussionsområden, som lämpligtvis borde föra diskussionerna framåt. Till varje diskussionsområde definierade vi dessutom ett antal hjälpfrågor. Syftet med dessa hjälpfrågor var att fungera som hjälpmedel om diskussionsområdena inte räckte för att föra diskussionen framåt. För intervjuguider, se bilaga A-C.

Våra intervjuer tog olika form beroende på i vilken fas i undersökningen vi befann oss i. Vi bedömde att skillnaderna i dessa intervjuer motiverar en uppdelning enligt följande:

3.4.1 Den första intervjun

Den första intervjun var som nämnts öppen. Vi hade många undersökningsvariabler och strävade efter att uppgiftslämnaren skulle leda oss in på variabler som var relevanta för deras projekt. Detta eftersom vi antog att vi inte skulle finna avvikelser som berodde på alla våra variabler. Den första intervjun var inte beroende av en intervjuordning. Se bilaga A för intervjuguide till den första intervjun.

Dessutom antog vi att uppgiftslämnarna hade mer erfarenhet av utveckling än vi, men öppenheten grundade sig även i en stor osäkerhet kring vad vi skulle komma att fokusera på vid en nästföljande dokumentundersökning. Syftet med den första intervjun var således att komma fram till olika områden vi kunde fokusera på under dokumentundersökningen.

Den första intervjun kom således att baseras helt på vår sammanställning av teori, men hur mycket den kom att styra berodde helt på uppgiftslämnaren. Detta ledde till ett andra val; skulle avsikten med intervjun vara dold eller öppen? Jacobsen (2002) beskriver etik som en central faktor i detta val, framför allt om avsikten i intervjun är känslig. Vi misstänkte att vårt syfte med intervjun skulle kunna uppfattas som känslig för vissa utvecklare (kan ses som kritik till deras implementationer). Dock var vi helt öppna med vårt syfte, då vi ansåg att en öppen avsikt skulle leda till bättre och fler relevanta svar.

3.4.2 Efterföljande intervjuer

De efterföljande intervjuerna skedde efter att dokumentundersökningen utförts. Vi hade då hittat specifikt material vi kunde basera vår nästkommande intervjuguide på. I denna fas handlade det främst om att visa på upptäckta och relevanta designaktiviteter för att få en motivering till varför de utförts. Se bilagor B och C för intervjuguide till de efterföljande intervjuerna.

De efterföljande intervjuerna kom i större utsträckning att styras av oss eftersom vi ville undersöka bakgrunden till varför specifika designaktiviteter genomförts. Vi tillät dock en viss öppenhet, då nya relevanta designaktiviteter eventuellt kunde uppmärksammas.

Att genomföra flera intervjuer med samma undersökningsobjekt är en viktig del av den kvalitativa ansatsen. Om vi endast har ett fåtal undersökningsobjekt och är intresserade av en djup bild så bör flera intervjuer genomföras (Jacobsen, 2002). Vi anser inte att vår ansats är fullt kvalitativ, men syftet med flera intervjuer är applicerbart även på vår undersökning.

3.5 Dokumentundersökningar

För vår undersökning var det viktigt att ha i åtanke att dokumentundersökningen inte var isolerad från intervjuerna. Dokumentundersökningen kunde endast genomföras efter intervju (då relevanta fokusområden var uppmärksammade), samtidigt som nästföljande intervjuer inte kunde genomföras utan att en dokumentundersökning genomförts.

Vår dokumentundersökning skedde i form av granskning av observationsobjektens implementationer, det vill säga deras programmeringskod. Med oss från intervju hade vi ett antal områden vi kunde fokusera på. Efter dokumentundersökningen hade vi ett antal designaktiviteter vi kunde fokusera efterföljande intervju på.

Jacobsen (2002) hävdar att en av de främsta anledningarna med att genomföra en dokumentundersökning är att få veta vad människor faktiskt gjort. Här kan dokumentundersökningen ses som ett *reliabilitetskomplement* till föregående intervju; stämmer det som sagts under intervju och kan vi styrka det som sagts under intervju?

Många av Jacobsens (2002) argument för att dokumentundersökning skulle vara problematisk faller bort i vår undersökning. Framför allt grundar sig hans problematisering på *reliabiliteten* i sekundärdata. I vårt fall är syftet med dokumentet (koden) inte direkt att agera som anteckning för vad som hänt, utan snarare lösningen av vad som hänt och bör således ges högre tillförlitlighet än exempelvis en dagbok.

Vi har valt att inte återge någon kod i detalj för respektive implementation, då vi har blivit ombedda av företagen att inte göra detta. Att återge koden bidrar heller inte till vårt resultat - uppsatsen skulle bli för lång och svår att följa. Vi kommer däremot att referera till företeelser ur ett mer generellt perspektiv.

3.6 Genomförande av analys av intervju

Enligt Jacobsen (2002) resulterar semistrukturerade intervjuer i mycket data, något vi efter vår undersökning kan bekräfta. Jacobsen påstår att efter intervju bör den resulterande datan kategoriseras. Kategorierna ska således ta form efter intervjun. Vår undersökningsprocess har dock utformats så att kategorier, i form av sammanställning av teori, skapats före intervjuerna.

Vi är medvetna om att en fördel med en kvalitativ ansats är att upptäcka nya fenomen under undersökningens gång (Jacobsen, 2002). Dock ansåg vi att en fördefinierad kategorisering skulle underlätta vår undersökning. Vi har även valt att analysera de båda projekten parallellt med varandra, vilket är syftet med det stora projektet i vår undersökning.

Vår kategorisering av intervjuer har följt följande arbetsgång: (1) Transkribera intervju. (2) Markera delar av intervjun som kan vara relevanta för vårt resultat, grundat på vår teori. (3) Presentera kategorierna med tillhörande data. (4) Sortera data så att data från intervjupersonerna kan jämföras.

3.7 Genomförande av analys av dokumentundersökning

Analysen av dokumentundersökningen har följt en liknande arbetsgång som för analysen av intervjuer: (1) Från inledande intervju identifierade vi ett antal möjliga undersökningsområden. (2) Dessa områden undersöktes och jämfördes med teori. (3) Om data var relevant för vår teori, antecknades denna och låg till grund för utformningen av nästföljande intervju.

Analysen av dokumentundersökningen har således varit helt grundad på vår teori. Om bristfällig design upptäckts, men som inte är relevant för vår teori, har denna bortsetts ifrån. Som exempel har ett antal säkerhetsrelaterade problem identifierats under undersökningens gång, men eftersom dessa inte grundar sig i vår teori har vi valt att bortse från dessa. Anledningen till detta är att vi utan teori inte kan motivera att det faktiskt rör sig om bristfällig design.

3.8 Metodkritik

Värt att poängtera är att vår forskningsfråga samt en del av metoden förändrats under undersökningens gång. Detta eftersom vi under undersökningen upptäckte att ett av våra observationsobjekt inte föll inom ramarna för vad vi uppfattade som ett litet projekt. Vår undersökning tog således en mer komparativ form. Dock ligger fokus fortfarande på det lilla projektet, medan det stora projektet ger en kompletterande syn på god mjukvarudesign.

Vi förstår att generaliserbarheten av vår undersökning minskar i samband med att överförbarheten och jämförbarheten mellan projekten minskar. Dock anser vi att vårt resultat stärks, då vi kan påvisa avvikelser som inte endast är förekommande i små projekt. Dessutom anser vi att en något mer komparativ undersökning bidrar till ett mer nyanserat resultat.

Även om en objektivitet har eftersträvats genom hela undersökningen, vill vi poängtera att total objektivitet varit väldigt svårt att uppnå. Detta grundar sig redan i valet av problemområde, då vi valt ett område vi är intresserade av och ett område där vi på förhand hade en viss kritisk syn på att god design skulle vara det samma oberoende av studieobjekt.

Vidare är urvalet av teori präglat av vad vi anser är relevant för forskningsfrågan. Det finns tusentals artiklar och böcker om mjukvarudesign och vi kan inte bevisa att vårt urval av teori är representativt för all teori som finns.

Denna brist i urval av teori påverkar även vår dokumentundersökning och utformning av intervjuer. Detta eftersom teorin ligger till grund för såväl dokumentundersökning som intervjuer. Vidare har kategoriseringen och sorteringen av intervjudata utförts efter vad vi anser vara relevant för vår teori och därmed vår forskningsfråga. Vi anser att vår kategorisering av data inte är fullt *valid* och *reliabel*, men att liknande undersökningar kommer visa på snarlika resultat.

Värt att nämna är att informanterna vi valt till vår undersökning är personer som vi haft kontakt med tidigare. Hur mycket detta påverkar vår undersökning är svårt att uppskatta.

4 RESULTAT

4.1 Presentation av resultat

Vi har valt att presentera resultatet efter vår teorisammanställning. Det är sammanställningen som legat till grund för vår undersökning och därmed även vårt resultat. Vi har vidare valt att lägga ihop resultatet för de båda undersökningsobjekten, då läsaren lättare kan följa deras gemensamma/motsägande argument.

På liknande sätt har vi valt att lägga ihop undersökningsobjektens dokumentundersökning av dess implementation, samt dess efterföljande intervju. På så sätt kan läsaren enkelt se kopplingen mellan de olika undersökningsfasernas resultat.

För att förenkla för läsaren kommer citering till bilagorna att ske genom en kod. Exempelvis I1:F2, som då står för intervju 1 fråga 2.

4.2 Undersökningsobjekt

IT Visioner, som uteslutande utför små projekt, har gett oss tillgång att granska ett personsökningssystem. Det enda systemet egentligen gör är att skicka ett personnummer till en viss myndighet, och om personen följer ett visst kriterium så returneras personnumret. Systemet är något mer komplext, då det måste kommunicera med ett äldre externt system. Informanten från IT Visioner kommer löpande i resultatkapitlet att refereras till som Intervjuperson A.

Journal Digital AB har gett oss tillåtelse att granska ett journalhanteringssystem. Detta system har varit under utveckling i flera år. Vår informant för detta system är nytillsatt, vilket även de andra två utvecklarna är. Projektets komplexitet är hög, vilket ledde till en inskolningsperiod för utvecklarna på sex månader. Antalet klasser är flera hundra, de är indelade i flera moduler, samt utveckling och produktion sker fortlöpande med hjälp av uppdateringar. Informanten från Journal Digital AB kommer i resultatkapitlet att refereras till som Intervjuperson B.

4.3 Inledande intervjuer

4.3.1 Implementation

Funktionalitet

I det lilla projektet ansåg man att i små projekt spelar det mindre roll hur funktionaliteten uppfylls. Så länge kraven uppfylls spelar det mindre roll hur funktionaliteten designas. I vissa fall implementeras även extra funktionalitet, även om den inte görs tillgänglig för slutanvändaren.

> *”[...] och så blir det ju ’copy/paste’.. och då kanske man lägger till massa funktioner. Funktioner som inte är användbara, men om jag ska göra ett projekt [...] så sätter jag mig ju inte ner med ett blankt papper”* – Intervjuperson A, I1:F4

I det lilla projektet identifierades även problem som kan uppstå med denna implementation av funktionalitet. Det förklarades att denna lösning på funktionalitetskrav är beroende av hur bra den tidigare implementationen är utformad. Dessutom kan denna återanvändning i slutändan resultera i tidsförlust om flera liknande projekt kräver liknande lösningar.

> *”Gör man ’copy/paste’ så får man ju hoppas att man skrev bra kod förra gången. [...] Man hade kanske kunnat byggt en klass man hade kunnat återanvända. Kanske en webbtjänst egentligen.”* – Intervjuperson A, I1:F10

På frågan om huruvida iterering av implementation av vissa lösningar skulle leda till merarbete påstods det i det stora projektet att förhastade implementationer kan leda till merarbete i slutändan, vilket även resulterade i försvårad förståelse för gammal kod.

> *”För när man håller på med någonting så har man ju det färskt i huvudet. [...] Sen när man kommer tillbaka till det så tar det ju lite tid att få in det i huvudet igen.”* – Intervjuperson B, I2:F19

I nedanstående tabell presenteras påstådda medvetna avvikelser från vad som allmänt ses som god design kopplat till funktionalitet. Även respektive informants åsikt om motiv presenteras.

Tabell 4.1: Påstådda avvikelser inom funktionalitet

Studieobjekt	Medvetna avvikelser	Motiv
Lilla projektet	Återanvändning (copy/paste).	• Risk för brist i funktionalitet. • Sparad tid. • Överflödig funktionalitet.
Stora projektet	Ofullständig funktionalitet.	• Försvårad förståelse. • Merarbete.

Förståelse

Intervjuperson A i det lilla projektet hade starka åsikter om vad som påverkar förståelse. Det förklarades att förståelse har en stark koppling till projektens storlek; förståelse kan uppnås genom små medel i små projekt, medan implementationer i stora projekt kräver fler åtgärder för att uppnå en god förståelse. Framför allt kopplas denna åsikt till regler för

koddokumentation, där det lilla projektet inte hade samma behov av struktur som större projekt.

> *”Så att det är ju därför man måste ha reglerna, men då enbart för stora projekt. Det finns ju ingen annan praktisk funktion, förutom då att det är god sed att göra det.”* – Intervjuperson A, I1:F14

I det lilla projektet hade man erfarenhet av att det alltid slarvas med koddokumentation. Detta verkar främst vara fallet för större projekt, där behovet av koddokumentation är större. Dock påpekade han att överanvändning av dokumentation ej leder till några fördelar. Koddokumentation ansågs endast nödvändig för avvikande företeelser – enligt det lilla projektet behöver man inte dokumentera funktionalitet som är självtalande. Däremot kan det vara användbart att dokumentera funktionalitet så som avancerade beräkningar. (Intervjuperson A, I1:F6)

I det lilla projektet följdes inte helt objektorienteringens regler om att klasser ska vara talande och verklighetsförankrade. Det sades att framför allt verklighetsförankring är något man försöker följa, men att det inte alltid är så enkelt. Intervjuperson A menade att avvikelser får ske och att det endast handlar om sunt förnuft. Däremot ansåg man i det lilla projektet inte alltid att ett objekts funktionalitet måste vara verklighetsförankrade. Om extra funktionalitet kan implementeras för att automatisera ett objekts övergripande funktionalitet så görs det, även om förståelsen eventuellt skulle minska.

> *”Alltså, de saker jag brukar följa är det sista. Att de ska ha någon form av verklighetsförankring. [...] men det är inte alltid så lätt. [...] Om man tittar lite på det här med ’properties’ inne i en klass så kan man ju göra mycket där. Men de kanske säger att: ’Nej så där ska du inte göra. Du ska bara hämta och lämna här’. Men det tycker jag man kan göra.”* – Intervjuperson A, I1:F13

Även om man i det lilla projektet hade starka åsikter emot överambitiös dokumentering av kod, insåg man att ett impulsivt utvecklande kan leda till merarbete i slutändan. Man var av den åsikten att snabbkodning leder till snabbt fungerande funktionalitet, men att man ibland får gå tillbaka och rätta till eventuella misstag – något som med brist på koddokumentation försvårar förståelse. (Intervjuperson A, I1:F7)

I det stora projektet såg man vikten av ett ordentligt genomfört förarbete till designfasen, snarare än att kasta sig in i kodningen av lösningen. Man ansåg att förståelse för projektet inte endast var viktigt för utvecklarna, utan även alla projektets intressenter. Denna förståelse säkrade man genom inledande möten med alla intressenter inför designfasen, i vilken man på ett förståeligt sätt, även för kunden, ritade upp strukturen för designlösningen. (Intervjuperson B, I2:F3)

Intervjuperson B för det stora projektet förklarade vidare att förberedelser och dokumentering av kod är något man alltid strävar efter, men som inte alltid sker. Han kopplar detta till brist på tid, men att man då i slutändan får gå tillbaka och städa i koden.

> *”Ibland är det ju så jättebråttom, så bara kodar man så att det funkar. Men sen så har han ju som mål att han går tillbaks sen och.. Städar i koden så att säga.”* – Intervjuperson B, I2:F4

I det stora projektet var man av åsikten att det ibland får göras vissa avvikelser från vad som allmänt anses som god utveckling för att säkra en implementations funktionalitet. Lösningar som på pappret allmänt anses vara goda behöver nödvändigtvis inte vara de mest optimerade.

> *”[...] 1000 rader kod med väldigt avancerade beräkningar. [...] det tar kanske en tusendels sekund att exekvera. Men har man en 10 rader kod och man ska ner och läsa i databasen fyra gånger, så tar det ju mycket längre tid.”* – Intervjuperson B, I2:F11

På frågor kring användbarheten i verktyg som underlättar designförberedelser svarade Intervjuperson B att framför allt strukturbaserade verktyg så som UML är användbara, men att det finns en flexibilitet i vad man kan använda. Han upprepade även att designförberedelser är viktiga för att underlätta implementationen.

> *”Någonstans måste man ändå få ner det med det här strukturerade. Och kunna rita för sig själv så man fattar hur man ska koda det, men ändå kunna berätta tillbaka: ’Är det så här du har tänkt dig?’.”* – Intervjuperson B, I2:F20

I nedanstående tabell presenteras påstådda medvetna avvikelser från vad som allmänt ses som god design kopplat till förståelse under implementationsfasen. Även respektive informants åsikt om motiv presenteras.

Tabell 4.2: Påstådda avvikelser inom förståelse under implementation

Studieobjekt	Medvetna avvikelser	Motiv
Lilla projektet	Avsaknad av regler för koddokumentation.	• Sparad tid.
	Endast kommentera avvikelser i koden.	• Merarbete om man måste återvända till koden. • Sparad tid om man inte behöver återvända till koden.
	Ej alltid strikt talande och verklighetsförankrade objekt.	• Projektet kan gå vidare.
Stora projektet	Brist på koddokumentation.	• Sparad tid innan release.
	Funktionalitet före läsbarhet.	• Optimering. • Större kodstorlek.
	Ofullständig funktionalitet.	• Försvårad förståelse. • Merarbete i det långa loppet.

Storlek

Intervjuperson A påpekade att storleken på implementationen inte hade, vad han visste, någon påverkan på lösningens användning. Dock förklarade han att man strävar efter att skriva kompakt kod. Vidare svarade han att storleken på projekt ska bestämma hur mycket extra utrymme som ska ges till dokumentation.

> *”Jag tycker; ska det vara ett litet projekt så ska det vara lite dokumentation [...] Ska du göra ett jättestort projekt så ska du ha mycket dokumentation. [...] Det finns egentligen inget, vad jag vet i alla fall, som kompileringsmässigt skulle göra så att det tar längre tid.”* – Intervjuperson A, I1:F7

Intervjuperson B nämnde tidigare att viss avvägning mellan god sed och funktionalitet bör göras. Han nämnde att tusen rader programmeringskod ibland kan lösa ett problem bättre än tio rader databaskod.

I nedanstående tabell presenteras påstådd medveten avvikelse från vad som allmänt ses som god design kopplat till kodstorlek. Även informantens åsikt om motiv presenteras.

Tabell 4.3: Påstådda avvikelser inom storlek

Studieobjekt	Medvetna avvikelser	Motiv
Lilla projektet	-	-
Stora projektet	Funktionalitet före storlek.	• Optimering. • Större kodstorlek.

4.3.2 Testning

Modularitet

I det lilla projektet var man av åsikten att en hög modularitet tjänar sitt syfte i stora projekt, men att bryta ut funktionalitet i mindre projekt kan leda till merarbete. Det påpekades även att det finns starka åsikter från personer som anser att man alltid ska göra rätt, men att det inte tjänar sitt syfte för det lilla projektet. Man var dock medveten om att det tjänar sitt syfte i stora projekt likt google.com. (Intervjuperson A, I1:F6)

I det större projektet lade man större vikt på att bryta ut funktionalitet. Det påstods att brist på modularitet kan leda till att utvecklaren förlorar kontroll över koden, något som varit ett problem under 80-talet. (Intervjuperson B, I2:F6)

I nedanstående tabell presenteras påstådd medveten avvikelse från vad som allmänt ses som god design kopplat till modularitet under testning. Även informantens åsikt om motiv presenteras.

Tabell 4.4: Påstådda avvikelser inom modularitet under testning

Studieobjekt	Medvetna avvikelser	Motiv
Lilla projektet	Brist på modularitet.	• Sparad tid.
Stora projektet	-	-

4.3.3 Drift

Förståelse

När det kommer till förståelsen av redan skriven kod anser man i det lilla projektet inte ha några större bekymmer av detta, så länge det gäller egen skriven kod. När det kommer till andras kod påpekades det att det blir betydligt svårare för förståelsen, då man ofta inte vet hur andra personer har tänkt när de konstruerade koden.

> *”Så i och med att jag gjort det i min egen stil då, så vet jag kanske varför jag gjorde så. [...] Men när jag får kod ifrån ett annat projekt. Jättesvårt, jättesvårt. Ingen aning hur den personen har tänkt.”* – Intervjuperson A, I1:F6

När frågan ställdes huruvida kodstorleken var viktig att tänka på i samband med förståelse och drift, så blev svaret aningen osäkert. Man menade i det lilla projektet att kompakt kod underlättar för en själv då koden blir mer överskådlig för en själv. Men om en annan utvecklare skulle sätta sig in i koden kanske det inte skulle vara lika enkelt att förstå.

> *”Det är så att man vill att det ska vara så ’tight’ som möjligt. [...] Du trycker ju ihop något som du tycker är självklart, men som någon annan inte förstår. Hade man brutit ihop det i delar så får du ju en bättre överblick.”* – Intervjuperson A, I1:F11

I det lilla projektet kritiserades den objektorienterade teorin gällande dokumentering av kod. Det påpekades att i mindre projekt är behovet av dokumentation inte lika stort då man gör vad situationen kräver och inte är styrd av någon större organisation. Om det sedan skulle visa sig vara problem så är det inte några svårigheter att gå tillbaka och ändra. Som tidigare har nämnts förstår man koden bättre om man själv har skrivit den. (Intervjuperson A, I1:F7)

För det stora projektet ställdes frågan om det var svårt att sätta sig in i koden när de och deras konsulter tog över projektet. De var de väl medvetna om projektets storlek och hade förberett sig genom att studera och förstå koden ett antal månader i förväg.

> *”De satte sig några månader innan för att bara sätta sig in i koden. För att då få en uppfattning om hur lång tid saker och ting skulle ta att ändra och så vidare.”* – Intervjuperson B, I2:F5

Det nämndes sedan att även förståelsen var viktig för intervjuperson B när konsulterna i ett senare skede kommer att lämna projektet och han ska ta över den framtida driften. På så sätt slipper företaget köpa in nya programmera så fort ändringar av systemet ska implementeras. (Intervjuperson B, I2:F8)

I nedanstående tabell presenteras påstådda medvetna avvikelser från vad som allmänt ses som god design kopplat till förståelse under drift. Även respektive informants åsikt om motiv presenteras.

Tabell 4.5: Påstådda avvikelser inom förståelse under drift

Studieobjekt	Medvetna avvikelser	Motiv
Lilla projektet	Brist på koddokumentation.	• Merarbete efter release. • Svårt att förstå annan utvecklares kod.
	Väldigt kompakt kod.	• Lättare att förstå för en själv. • Svårare att förstå för andra.
Stora projektet	Brist på koddokumentation.	• Merarbete vid överlåtande av projekt.

Modularitet

Av erfarenhet för Intervjuperson A hade många projekt en tendens att bli mer omfattande än vad de i början troddes bli. Det här gjorde att koden ofta blev rörig framåt slutet av utvecklingsprocessen, då det från början inte var tänkt att bli ett så stort projekt. Trots detta lade man oftast inte upp någon plan för projektet i förväg, utan det blev istället att koden byggdes på allteftersom nya krav dök upp.

> *”Många projekt börjar med att man gör något litet och sen växer och växer och växer, för folk är intresserade och vill ha mer. Och då blir det ju naturligt att man slänger ihop kod.”* – Intervjuperson A, I1:F3

Vidare menade Intervjuperson A att det ofta blev att ett projekt började som ett lite mindre, mer hanterbart projekt, men att det växte med tiden allteftersom kunden kom med nya krav på programvaran. Det kunde leda till att man ofta låste in sig på ett spår, då man valde att helt bygga vidare på sin nuvarande kod, vilken var strukturerad efter hur projektet var från början. På så sätt blev det problematiskt i framtiden när man eventuellt beslutade sig för att ändra grundstrukturen på projektet. Intervjuperson A gav exempel på detta, då han en gång lyckats förändra grundstrukturen i tid, vilket inte varit möjligt senare i projektet. (Intervjuperson A, I1:F6)

I det lilla projektet trodde man att det skulle blir enklare om man utgick från en bra stuktur när man började projekt. Man förklarade att ju mer väldefinierat något är desto fler beroenden blir det mellan klasser och funktioner, vilket ansågs vara en nackdel. Men man ansåg även att det borde bli lättare om man definierar en komponent väl från början, men att man inte hade någon erfarenhet av detta.

> *”Både jag och nej för att.. Om du ska byta en del så är den ofta kopplad till massa andra funktioner även om den är väldigt väldefinierad och har bra interface.”* – Intervjuperson A, I1:F8

Ytterligare problem som kunde uppdagas i driften var relaterat till statisk information, som ibland kunde förekomma. Det kunde ställa till stora problem då informationen ändrades och man i efterhand behövde ändra i koden. Därför ansåg Intervjuperson A i det lilla projektet att

vikten av att lyfta ut sådan information, exempelvis till xml-filer, var oerhört stor. Det förenklade efterarbetet med driften då man slapp gå igenom all kod igen.

> *”Man sätter ihop det, slänger ut det, och sen är det färdigt så att säga. Och sen så när det är klart, så ska man plötsligt flytta allt det här till en annan server. Oh shit! Tänker jag ju då. Nu ska jag in där och leta upp alla ’connect strings’ jag hade överallt.”* – Intervjuperson A, I1:F12

I det stora projektet lades större vikt vid modularitet, då deras arbete främst handlade om ett stort projekt med många klasser att övervaka. Det nämndes att deras konsult, som huvudsakligen stod för den tekniska biten av utvecklandet, hade erfarenhet av projekt som blivit okontrollerbara då de inte följt en lämplig struktur. Det påstods även att de kontinuerligt försöker hålla koden ren och inte blanda in för många metoder, som exempelvis överdrivet användande av databashanteringsmetoder. Detta bidrog på sikt till att systemet blev mer lätthanterligt.

> *”Janne har ju varit med sen 80-talet och kodat så han har ju varit inne i det här träsket att man bara kodar på och till slut tappar man kontrollen över koden. Så han, han tycker det är oerhört viktigt att göra saker rent. [...] Anledning till att han gör det är ju att det ska bli lättare att underhålla systemet senare.”* – Intervjuperson B, I2:F6

Vidare nämndes det i det stora projektet att deras drift förenklas märkbart på grund av att deras system är webbaserat. Uppdateringar kan därför utföras oftare än med många andra projekt, även om de måste stänga ner sina system en timme under uppdateringen. (Intervjuperson B, I2:F14)

I nedanstående tabell presenteras påstådd medveten avvikelse från vad som allmänt ses som god design kopplat till modularitet under drift. Även informantens åsikt om motiv presenteras.

Tabell 4.6: Påstådda avvikelser inom modularitet under drift

Studieobjekt	Medvetna avvikelser	Motiv
Lilla projektet	Brist på modularitet.	• Brist på föränderlighet. • Förlorad kontroll då projektet växer. • Merarbete. • Svårare att byta ut moduler.
Stora projektet	-	-

Säkerhet

Ingen av intervjuerna påvisade något som kunde härledas till vår karakterisering av säkerhet – att säkerhet påverkas av djupet i arvshierarkier.

4.4 Dokumentundersökningar och nästföljande intervjuer

4.4.1 Implementation

Funktionalitet

Dokumentundersökningen för det lilla projektet visade att all funktionalitet implementerats i en enda klass. Denna klass skötte databaskommunikation, filrelaterade funktioner samt generell logik. På frågan varför man valt att implementera funktionaliteten enligt ovan svarades det att man hade vetskap om dess betydelse, men att det i detta lilla projekt inte hade någon prestandamässig betydelse. (Intervjuperson A, I3:F3)

Vidare visade dokumentundersökningen för det lilla projektet att viss prioritet lagts på funktionalitet snarare än sammanhållning, då viss funktionalitet implementerats i property-funktionerna av en klass. Intervjuperson A motiverade detta med att det inte endast handlar om att tjäna tid, utan framför allt för att automatisera viss funktionalitet. Detta i jämförelse med att utföra denna funktionalitet vid varje funktionsanrop. (Intervjuperson A, I3:F12)

Likt det lilla projektet, visade dokumentundersökningen för det större projektet på att i många instansvariablers property-funktioner, valde utvecklarna att involvera fler operationer än att bara hämta och lämna värden. Som motivering till varför de valde att utforma systemet på detta sätt sades det:

> *”Även om det inte direkt är en variabel så tycker han att man lägger det som en property ändå.”* – Intervjuperson B, I4:F10

I nedanstående tabell presenteras observerade medvetna avvikelser från vad som allmänt ses som god design kopplat till funktionalitet. Även respektive informants åsikt om motiv presenteras.

Tabell 4.7: Observerade och motiverade avvikelser inom funktionalitet

Studieobjekt	Medvetna avvikelser	Motiv
Lilla projektet	Låg sammanhållning.	• Sparad tid.
	Brist på inkapsling.	• Automatisering av funktionalitet. • Lägre sammanhållning.
Stora projektet	Brist på inkapsling.	• Automatisering av funktionalitet. • Lägre sammanhållning.

Förståelse

Dokumentundersökningen för det lilla projektet visade att någon namngivningsstandard för form-kontroller inte använts. Exempelvis fanns det en knapp med namnet ”button1”. Intervjuperson A motiverade detta med att projektets storlek inte motiverade en namngivningsstandard:

> *”Det är ju så enkelt att se liksom, vilket ’event’ som tillhör liksom. Man behöver inte köra någon ’naming-standard’ rakt igenom liksom.”* – Intervjuperson A, I3:F4

Vidare visade dokumentundersökningen för det lilla projektet på ett sparsamt användande av koddokumentation. Dock visade undersökningen även att man hade dokumenterat funktioner som redan hade en klar innebörd. Han förklarade att dokumentation tar olika form beroende på utvecklarens stil, men att man egentligen bara ska dokumentera saker som är avvikande. Intervjuperson A ansåg sig vara klar då han implementerat funktionaliteten, även om koddokumentation inte skrivits. (Intervjuperson A, I3:F10)

Något som kunde uppmärksammas i dokumentundersökningen för det stora projektet var att koden hade mycket utförlig dokumentation på sina ställen. Man motiverade den goda koddokumentationen med att man ökade förståelsen för olika typer av funktioner. (Intervjuperson B, I4:F19)

Även följande motivation till god koddokumentation gavs:

> *”Alltså, jag tror att han tänker att när han letar efter en metod så vet han ju i vilken region han ska leta i. Så blir det ju bara en femtedel så många att titta igenom. Sen är det ju också så att om det är någon annan som ska koda, så är det ju bra att veta var man ska leta.”* – Intervjuperson B, I4:F20

I nedanstående tabell presenteras observerade medvetna avvikelser från vad som allmänt ses som god design kopplat till förståelse under implementation. Även informantens åsikt om motiv presenteras.

Tabell 4.8: Observerade och motiverade avvikelser inom förståelse under implementation

Studieobjekt	Medvetna avvikelser	Motiv
Lilla projektet	Ingen namngivningsstandard.	• Sparad tid.
	Brist på koddokumentation.	• Sparad tid.
Stora projektet	-	-

Storlek

Under dokumentundersökningen för det lilla projektet observerades att mycket av koden var redundant. Intervjuperson A bekräftade att mycket ’copy/paste’ använts och motiverade detta med:

> *”Det är mycket ’copy/paste’. [...] Sen ’copy/pastar’ du den funktionen om och igen. Det är ett litet hastverk kan man säga. Men om du gör ett projekt i den storleksklassen så.. Jag vet inte vad det riktigt gör för skillnad egentligen.”* – Intervjuperson A, I3:F1

4.4.2 Testning

Modularitet

Vi fann i dokumentundersökningen för det större projektet att på flera ställen i koden fanns vitala delar som mycket väl kunde brytas ut för att göra koden mer dynamisk. När vi i intervjun diskuterade detta så ledde diskussionen in på hur de upptäcker sådana fel i systemet. Som svar på frågan hur de gick tillväga fick vi:

> *”Då låter vi systemet gå och sen letar vi fel via testsidan då, som inte berör de som köper tjänsten då. Så letar vi upp felet där och korrigerar det. Sen så stänger man bara ner systemet en timme och så lyfter man in en ny version.”* – Intervjuperson B, I4:F17

4.4.3 Drift

Förståelse

Att det lilla projektet inte använde någon namngivningsstandard och att det inte ansågs finna någon anledning till att göra det i så litet projekt, har tagits upp i delkapitel 4.4.1.

Koddokumentation diskuterades med Intervjuperson A för det lilla projektet. Han svarade att koddokumentationen är resultatet av utvecklarens personliga stil, och att endast avvikande saker behöver dokumenteras. Detta togs upp i delkapitel 4.4.1.

Dokumentundersökningen för det lilla projektet visade att all felhantering skedde med hjälp av returnering av 1:or och 0:or; fungerar det skickas en nolla tillbaka, annars en etta. Intervjuperson A ansåg inte att en mer detaljerad felhantering behövdes, då användaren av systemet ändå inte hade förstått ett mer detaljerat felmeddelande. Att det var god sed motiverade inte en förändring av felhanteringen. (Intervjuperson A, I3:F16)

Intervjuperson A påpekade dock att felhantering är viktigt och att mer tid lagts på detta om projektet varit större:

> *”Men.. Hade jag haft ett lite större program som skulle användas av 50.000 människor så hade jag ju självklart lagt ner enormt tid på att folk ska förstå vad felet är och man skulle haft en kundtjänst som tog emot felet då va. Man kanske även skulle haft en kod då va, så att man kunde slå upp vad felet är..”* – Intervjuperson A, I3:F16

Dokumentundersökningen för det större projektet visade att på många ställen i koden saknades implementation för felhantering. Om systemet av någon anledning skulle sluta fungera så fanns inte funktionalitet nog till att ge användarna tillräckligt med information för att arbeta på som vanligt. När vi ställde Intervjuperson B frågan om varför det var utformat på det här sättet fick vi följande som svar:

> *”[...] de har själva bestämt sig för att inte ha en massa ’try and catch’. Därför att de resonerar som så att om det smäller så är det bättre att det smäller rejält. [...] För har man en massa ’catch’ så att folk fortsätter att jobba i systemet trots att det inte fungerar, så kan det ta mycket längre tid att få reda på det.”* – Intervjuperson B, I4:F22

I nedanstående tabell presenteras observerade medvetna avvikelser från vad som allmänt ses som god design kopplat till förståelse under drift. Även respektive informants åsikt om motiv presenteras.

Tabell 4.9: Observerade och motiverade avvikelser inom förståelse under drift

Studieobjekt	Medvetna avvikelser	Motiv
Lilla projektet	Ingen namngivningsstandard.	-
	Brist på koddokumentation.	• Merarbete.
	Brist på talande felhantering.	-
Stora projektet	Avsaknad av felhantering.	• Säkerställning av datans integritet. • Ökad förståelse för att fel inträffat.

Modularitet

Att det lilla projektet innehöll mycket ’copy/paste’ tas upp i såväl detta delkapitel som delkapitel 4.4.1. På frågan om det ansågs att projektet hade något behov av utbrytbarhet svarades:

> *”Absolut inte. [...] Sen att det blev så komplicerat är bara för att de har ett så gammalmodigt system.”* – Intervjuperson A, I3:F3

Något som uppdagades i dokumentundersökningen för det större projektet var att i många ställen av koden kunde man hitta upprepningar av funktioner. Detta i sin tur ledde till att sammanhållningen försvagades. Som svar på huruvida detta var medvetna designbeslut från deras sida svarade de att de försöker lyfta ner logik från olika ställen. Detta eftersom man upptäckt att en del av logiken ligger på fyra olika ställen. (Intervjuperson B, I4:F8)

Relevant för drift, som upptäcktes under observationen för det lilla projektet, är att funktionalitet implementeras i ”property”-funktioner. Mer om detta i delkapitel 4.4.1. På frågan om varför man inte brutit ut denna funktionalitet till egna funktioner fick vi till svar:

> *”Jag vet inte riktigt vad jag skulle tjäna.. Vad skulle jag tjäna på det. Vad säger egentligen boken om vad jag skulle tjäna på det?”* – Intervjuperson A, I3:F13

Dokumentundersökningen för det lilla projektet visade att viss utbrytbarhet gällande ”file locations” gjorts, men att de inte använts fullt ut. Detta menade dock Intervjuperson A berodde på slarv från hans sida och att man medvetet försöker bryta ut sådan information. (Intervjuperson A, I3:F14)

Dokumentundersökningen för det större projektet visade på att mönsterlösningar inte var något som frekvent användes. Utvecklarna var oeniga om huruvida mönsterlösningar skulle implementeras eller inte. Till slut valde de att inte implementera mönsterlösningar, då det i deras utvecklingsmiljö fanns inbyggd funktionalitet för att automatisera en stor del av funktionaliteten, vilka även abstraherar en del av logiken. (Intervjuperson B, I4:F4)

Vidare på samma område kunde vi på vissa ställen i dokumentundersökningen av det större projektet finna att komponenter flitigt bröts ut. Vi frågade därför om detta gjordes för att slippa upprepningar och problem. Som svar fick vi följande:

> *”[...] i och med att så mycket var hårdkodat innan så var det väldigt ovänligt för förändring. [...] Nu när Janne bygger det mer så att man ska kunna modifiera det efter kundens önskemål. Då är det mycket lättare att gå ut och träffa kunder.” –* Intervjuperson B, I4:F24

I nedanstående tabell presenteras observerade medvetna avvikelser från vad som allmänt ses som god design kopplat till modularitet under drift. Även respektive informants åsikt om motiv presenteras.

Tabell 4.10: Observerade och motiverade avvikelser inom modularitet under drift

Studieobjekt	Medvetna avvikelser	Motiv
Lilla projektet	Brist på inkapsling.	• Lägre sammanhållning.
	Brist på modularitet.	-
Stora projektet	Avsaknad av mönsterlösningar.	• Högre automatisering. • Lägre abstraktion.

Säkerhet

Ingen av intervjuerna eller dokumentundersökningarna påvisade något som kunde härledas till vår definition av säkerhet – att säkerhet påverkas av djupet i arvshierarkier.

5 ANALYS & DISKUSSION

Med hjälp av vårt teoretiska ramverk har vi analyserat empirin. Vi ville komma fram till vilka medvetna avvikelser små projekt gör från god design och vilka motiv som ligger bakom dessa designval. Vi presenterar i detta kapitel även viss data som inte faller inom ramen för vårt teoretiska ramverk, men som vi anser hjälper till att förklara motiv bakom avvikelser från god objektorienterad design.

Intervjupersonerna kommer likt föregående kapitel att refereras till som Intervjuperson A och Intervjuperson B. Citeringen till bilagor kommer även den likt föregående kapitel att ske genom en viss kod. Exempelvis I1:F2, som då står för intervju 1 fråga 2.

5.1 Implementation

5.1.1 Funktionalitet

Intervjuperson A i det lilla projektet menade att så länge systemet löser sin funktion så spelar det mindre roll hur lösningen ser ut på insidan. Detta visade sig även i dokumentundersökningen, där mycket av systemets funktionalitet var uppbyggt efter ”copy/paste” snarare än vad som allmänt ses som god objektorienterad design (delkapitel 2.7). Intervjuperson A motiverade detta med att systemet klarar av att utföra det som det är designat för att utföra, men framför allt vinsten i tid denna princip medför. Intervjuperson B i det större projektet hade en liknande inställning, nämligen att man ibland får göra vissa avvikelser från vad som anses vara god design för att säkra en produkts funktionalitet.

Viss koppling mellan empiri och teori finns. Budgen (1995) skrev att den ultimata designegenskapen är huruvida ett systems funktionalitet löser det problem det är designat för. Dock hävdar Britton & Doake (2005), Budgen (1995) och Soni et al. (2009) att entiteter endast ska utföra saker som är relevanta för dem själva. I fallet med det lilla projektet ledde ”copy/paste” till merfunktioner, vilket strider mot god design.

I det stora projektet påpekades att om funktionalitet inte implementeras korrekt från början så kan detta leda till problem senare. I det lilla projektet var man av samma åsikt, men att det främst rör projekt som är stora. Det påpekades i det lilla projektet även att det inte alltid går att förutse storleken på ett projekt - att ett litet projekt som implementerats slarvigt, kan bli svårt att bygga ut om behov finns.

Att begränsa funktionalitet till en klass sammanhållning visade det sig i båda projekten att det inte alltid är nödvändigt. Det visade sig under såväl intervjuer som dokumentundersökningar, för såväl det lilla som stora projektet, att manipulation av en klass attribut direkt i instansmetoder leder till sparad tid och kodstorlek. Inga av informanterna såg några nackdelar med detta. Budgen (1995) påpekar dock att en god sammanhållning kan användas som kriterium för god design. Britton & Doake (2005) samt Budgen (1995) hävdar även att inkapsling är ett kriterium för god design.

5.1.2 Förståelse

I det lilla projektet var Intervjuperson A av åsikten att förståelse är relativt till projektets storlek. Att en hög förståelse kan nås med små medel i små projekt. Detta visade sig även i dokumentundersökningen, där koddokumentation var frånvarande och objekt ej hade talande namn. Detta var dock inget som ansågs påverka förståelsen negativt, då det ansågs att respektive utvecklare själv förstod ens egen kod. Dock påpekade Intervjuperson A att han strävar efter talande klassnamn.

I det stora projektet påpekades att de strävar efter att ha god dokumentation i koden. I dokumentundersökningen visade det sig att god koddokumentation existerade på de flesta ställena, men det påpekades att detta var tidskrävande att dokumentera, men att man samtidigt sparade tid genom att koden blir mer lättöverskådlig.

Informanterna hade skilda åsikter angående vikten av designförberedelser. I det stora projektet ansåg man att goda designförberedelser är viktiga för att underlätta implementationen, medan Intervjuperson A i det lilla projektet påpekade att detta var tidskrävande och att han oftast sätter sig ner och direkt börjar med implementationen.

Förståelse beskriver hur lätt en design kan tolkas och implementeras Zhu (2005), särskilt om ny personal tillsätts till projekten. Kanske är det då inte särskilt konstigt att intervjupersonerna är av skilda åsikter kring dess betydelse, med tanke på deras respektive projekts skilda storlek.

5.1.3 Storlek

Båda informanterna ansåg att storleken på en lösning inte hade någon direkt koppling till lösningens kvalitet. I det lilla projektet sades det att man endast använde storlek som en personlig utmaning, men att man inte kände till något om att det skulle påverka lösningens prestanda. Intervjuperson B i det stora projektet påpekade att ibland kan 1000 rader kod lösa ett problem mer effektivt än tio rader databaskod.

Som nämnts tidigare visade dokumentundersökningen för det lilla projektet att koden hade kunnat göras mindre. Dock påpekades det att funktionaliteten då inte hade förbättrats och att det endast lett till ökad utvecklingstid.

I teorin har vi presenterat storlek som ett kvalitetsmått (Chatzigeorgiou at el., 2008). Inga av våra informanter anser att storlek har någonting att göra med en designs kvalitet. Vår undersökning är ej tillräckligt omfattande för att kunna bortse från storlek som ett kvaltietsmått.

5.2 Testning

5.2.1 Modularitet

Båda informanterna höll med om att en hög modularitet är viktigt för större projekt. Dock påpekades det i det lilla projektet att utbrytning av funktionalitet i mindre projekt kan leda till merarbete. Kontrasterna projekten emellan var väldigt höga, där det lilla projektet hade en väldigt låg modularitet, medan det stora projektet hade en väldigt hög modularitet.

Intervjuperson B i det stora projektet motiverade den höga modulariteten med att man med en låg modularitet lätt kan tappa kontrollen över koden.

Dock visade dokumentundersökningen för det stora projektet att vissa delar hade kunnat brytas ut mer. Man tyckte inte att detta var ett problem, trots att de vid körningsfel stänger ner systemet en timme för att rätta till problemen som uppstår.

Här bekräftar empirin vår teori (Witt et al. enligt Zhu, 2005)(Britton & Doake, 2005). Visserligen använde sig inte det lilla projektet av hög modularitet, men man var införstådd med vilka konsekvenser detta kunde generera. Skilt från teorin är däremot uppfattningen att modularitet kan leda till merarbete.

5.3 Drift

5.3.1 Förståelse

Båda informanterna ansåg att det är svårt att förstå en annan utvecklares kod. I det lilla projektet ansågs detta inte vara lika problematiskt, där man efter ett tag lär sig hur sina projektmedarbetare utvecklar. Intervjuperson B i det stora projektet, som tog över efter att tidigare utvecklare fått sluta, fick lägga månaders förberedelser för att förstå vad koden egentligen gjorde innan han kunde starta nyutvecklingen av systemet.

I det lilla projektet var Intervjuperson A av den åsikten att den teori som existerar inom objektorienterad utveckling inte stämmer överens för små projekt. Han påpekade att det i små projekt är mycket lättare än i större projekt att rätta till fel i efterhand. Han var av den uppfattningen att endast avvikande företeelser behöver dokumenteras. Intervjuperson B i det stora projektet lade mycket större vikt på koddokumentation relaterat till drift eftersom han en dag skulle ta över projektet. Som nämnts tidigare skiljer sig, efter utförd dokumentundersökning, de båda projektens användande av koddokumentation mycket åt.

I det lilla projektet påpekade Intervjuperson A att inte endast koddokumentation kan öka förståelsen för en implementation. Han menar att om kodstorleken hålls nere så kan kompaktheten underlätta för en själv, men att den samtidigt kan försvåra för utomstående utvecklare.

Som nämnts tidigare (delkapitel 5.1.2) anser Intervjuperson A i det lilla projektet inte att namngivningsstandarder underlättar förståelse särskilt mycket i små projekt. Under dokumentundersökningen visade det sig att det, enligt god design, fanns namngivningsbrister.

Dokumentundersökningen för det lilla projektet visade att felhanteringen ej var talande. Detta designbeslut motiverades med att slutanvändaren ändå inte hade förstått något annat felmeddelande, samt att projektets storlek inte motiverade det. Dock påpekades att om projektet varit större hade mer tid lagts på felhantering. Efter dokumentundersökningen för det stora projektet identifierades en avsaknad av felhantering. Detta motiverades med att förståelsen visserligen minskar, men att systemets integritet kan säkras.

Som nämnts tidigare (delkapitel 5.1.2) har informanterna olika syn på förståelse. Intervjuperson A ansåg inte att förståelse var lika viktigt för små projekt och var direkt kritisk mot vad allmän teori förespråkar. Detta skiljer sig från vad Zhu (2005) hävdar; man ska sträva

efter en hög förståelse. Detta gällde såväl koddokumentation som namngivningsstandarder i det lilla projektet, vilka enligt Intervjuperson A mer lämpade sig för större projekt.

5.3.2 Modularitet

Intervjuperson A i det lilla projektet påpekade att en låg modularitet kan försvåra utbyggnaden av ett system och att designförberedelser kan underlätta detta. Motsägelsefullt sade han även att en bra grundstruktur skapar fler beroende mellan klasser och funktioner. Intervjuperson A ansåg att information som i efterhand kan komma att ändras bör brytas ut så mycket som möjligt för att underlätta driften. Dokumentundersökningen för det lilla projektet visade att statisk information bröts ut, men att denna sällan användes. Intervjuperson B i det stora projektet var av liknande uppfattning; statisk information bör brytas ut. Detta visade sig även stämma i dokumentundersökningen för det stora projektet.

Intervjuperson B i det stora projektet lade väldigt stor vikt vid hög modularitet, då han ansåg att projekt med låg modularitet kan leda till förlorad kontroll av programkod. Dokumentundersökningen visade att stor vikt lagts på utbrytbarhet. Vidare påpekades att deras drift underlättats väldigt mycket, då systemets funktionalitet är centralt utbrutet på webben.

Till skillnad från det stora projektet, ansåg inte Intervjuperson A att hans projekt var i behov av utbrytbarhet. Detta på grund av att systemet i grund och botten bara gjorde en enda sak. Dock visade dokumentundersökningen för det stora projektet att funktioner upprepades och därmed försvagades sammanhållningen. Intervjuperson B var dock medveten om detta och sade att de försöker bryta ut sådan funktionalitet så mycket som möjligt.

Som nämnts i delkapitel 5.1.1 var båda informanterna av den åsikten att de inte såg några nackdelar kopplat till implementation med att manipulera en klass attribut direkt i instansmetoderna. I det lilla projektet kritiserades det objektorienterade tankesättet och man förstod inte vad som skulle tjänas på att bryta ut denna funktionalitet.

Ingen av projekten hade implementerat någon mönsterlösning bortsett från de automatiskt genererade mönsterlösningarna i respektive utvecklingsmiljö. Dokumentundersökningen för det stora projektet visade ett typexempel där en viss mönsterlösning hade kunnat användas. Detta motiverades med att det rådde delade meningar i utvecklingslaget om dess nytta.

Här kan två huvudanledningar med hög modularitet identifieras. För det första är båda informanterna eniga om att man ska försöka bryta ut statisk information så mycket som möjligt för att underlätta drift. Vidare använde det stora projektet modularitet som ett verktyg för att strukturera upp implementationen av projektet. Detta behov fanns inte i det lilla projektet. Båda informanterna förstod vikten av modularitet, vilket är förenligt med Witt et al. enligt Zhu (2005) och Britton & Doake (2005). Däremot, till skillnad från teorin, påpekades i det lilla projektet att alltför hög modularitet kan leda till fler beroenden. Dessutom sades det att det tar tid att bryta ut allt.

5.3.3 Säkerhet

Vår undersökning resulterade inte i relevant data gällande säkerhet. Vi upptäckte säkerhetsrelaterade brister, men eftersom vi inte kunde knyta dessa brister till teori valde vi att

bortse från dessa. Vi utesluter inte att djupet av arvshierarkier kan användas som kvalitetsmått, men för vår undersökning var det inte applicerbart.

5.4 Motiv bakom avvikelser

Vi har med vår undersökning, resultat och analys presenterat ett antal avvikelser som existerar för vårt undersökningsobjekt. Fokus har legat på avvikelserna i sig, även om motiveringar bakom dessa avvikelser har presenterats. I detta delkapitel läggs därför mer fokus på motiveringarna bakom avvikelserna. Vi har flera motiv, vilka vi valt att kategorisera enligt följande underkapitel:

5.4.1 Resurser & krav

> *”Det enda som egentligen styr det i slutändan är ju pengar. Hur mycket pengar är företagen, både internt och externt, villiga att lägga på de här projekten?”* – Intervjuperson A i det lilla projektet, I1:F6

Det har efter undersökningen blivit tydligt att det primära motivet bakom medvetna avvikelser, från definitionen av god design, är att spara resurser (då främst tid). Intervjuperson A i det lilla projektet har genomgående i vår undersökning visat att han tagit flera genvägar för att spara tid. Uttryck så som att: ”Varför ska man återuppfinna hjulet?” verkar ligga till grund för många av de designval som tagits.

Att återanvända lösningar har visat sig vara vanligt förekommande i det lilla projektet – detta för att spara tid, även om den återanvända lösningen besitter viss extra funktionalitet som inte används. Med motiveringen ”sparad tid” struntades det även i viss koddokumentation i det lilla projektet. Man menade att sådant endast tar tid och att han förstår sin egen kod och lär sig förstå andra utvecklare i sin nära omgivning. Intervjuperson B i det stora projektet hade däremot lärt sig vikten av dokumentation – delvis på grund av dålig dokumentation hade det nytillsatta utvecklingslaget behövt lägga månader på att sätta sig in i det nya projektet. På så vis hade brist i dokumentation lett till merarbete.

> *”Även om det inte är deadlines så.. så skapar han ju en stress ibland, som gör att det blir det här snabba sättet att utveckla på ändå. Alltså, att det att man lovar saker till kund.”* – Intervjuperson B i det stora projektet, I2:F18

Vi anser att resurser och krav hör ihop, då en kravspecifikation existerar och någon måste bedöma vilka resurser som ska sättas in för att tillfredställa denna. Enligt citatet ovan förstod Intervjuperson B i det stora projektet att kravet om leverans påverkade utvecklingen. Utvecklarna i det stora projektet hann inte implementera allt eftersom de hade ett krav på att leverera vid ett visst datum.

Intervjuperson A är mer van vid mindre projekt, vilka rimligtvis bör ha en mindre kravspecifikation. Efter undersökningen har det visat sig att motivet bakom flera av designvalen i det lilla projektet varit ”funktionalitet före kvalitet” – så länge användarna kan göra det de krävt kunna göra så spelar det mindre roll hur implementationen ser ut under ytan.

5.4.2 Projektets föränderlighet

> *”Lite så är det kanske. Det blir ofta att man börjar på ett sätt för att göra något snabbt.. och sen blir det kardinalfel. Och när man kommer den här långa resan som går i ett projekt, så blir det att det blir ett hopkok.”* – Intervjuperson A i det lilla projektet, I1:F4

Vi identifierade och visade tidigt att uppsatsen mycket skulle komma att handla om balansgången mellan implementation och drift, vilket vi visade genom vår kategorisering av vårt teoretiska ramverk (delkapitel 2.9). Kräver projektet ständig förändring efter release så ökar även kravet på god design. Som exempel har det stora projektet varit under utveckling i flera år, men vidareutvecklas fortfarande trots att produkten säljs och används. Denna föränderlighet ställde, enligt vår undersökning, därför en del krav på såväl modularitet som förståelse.

Det mindre projektet har, enligt Intervjuperson A, en tydligare slutpunkt. Det talades om såväl ”throw-away-code” som okunniga slutanvändare, vilket ställde mindre krav på modularitet och förståelse. Däremot hade Intervjuperson A viss erfarenhet av projekt som vuxit okontrollerat, men Intervjuperson A visar i vår undersökning att små projekt kräver mindre efterarbete.

5.4.3 Utvecklarens personlighet & historia

> *”Jag tror utveckling blir lite hur man är som person liksom. Jag är nog väldigt energisk. Jag vill ha saker gjorda väldigt fort.”* – Intervjuperson A i det lilla projektet, I3:F5

Det har i vår undersökning visat sig att utvecklarens personliga egenskaper påverkar om avvikelser från definitionen av god design uppstår eller ej. Intervjuperson A identifierade sig själv, enligt ovan citat, som en väldigt energisk person som vill få saker gjorda fort. Vidare i undersökningen nämnde Intervjuperson A att när han skapat en funktion så anser han sig vara klar med denna. Vad han då glömt är koddokumentation och kodoptimering.

> *”Och jag kan tycka att det är lite jobbigt att den känslan att nya grejer som ska utvecklas fyller på snabbare än man hinner göra färdigt. För mig är det en ny grej eftersom jag kommer från kommunvärlden. Men Janne som har utvecklat, han är cool lugn.”* – Intervjuperson B i det stora projektet, I2:F18

Vi anser att personliga egenskaper inte går att bortse ifrån. Utifrån vår erfarenhet av mjukvarudesign har det visat sig att vissa lägger större vikt vid struktur än hastighet, och vissa lägger mer vikt vid felhantering än andra. På samma sätt förklarade Intervjuperson A att vissa gillar att dokumentera, medan andra inte gör det. Intervjuperson B nämnde att en av hans utvecklare är väldigt noggrann som person, något som enligt Intervjuperson B påverkar denne utvecklares sätt att arbeta.

> *”Allting måste bli mer effektivt och då har man inte tid med traditionell utveckling som har sina rötter i 80-talet skulle jag säga. Där de utvecklade på ett helt annat sätt. De hade gott om tid, var noggranna, hade ont om minne, ont om disk.”* – Intervjuperson A i det lilla projektet, I1:F16

Även erfarenheter och personlig professionell historia kan påverka vilken syn man har på avvikelser från god design. Intervjuperson A i det lilla projektet nämnde att de som utvecklat under 80-talet är noggrannare med att saker görs enligt god designs regler. Intervjuperson A, som uppskattningsvis inte fastnat i detta historiska tankesätt, gör vissa avvikelser från god design som inte hade gjorts förr; som exempel berättade han att han av lathet kan tillåta ett personnummer att hålla mer än tio siffror eftersom den lilla mängden extraminne är försumbart. Vidare berättade Intervjuperson B att han föreslagit mönsterlösningar till den mer erfarne utvecklaren i det stora projektet, men att denne utvecklare valt att inte implementera dem.

6 SLUTSATS

Vi har i denna studie identifierat några kriterier för kvaliteter för god design och tagit reda på vilka designaktiviteter som leder till dessa. Vi har utifrån dessa kriterier undersökt två projekt, ett litet och ett större, och lyckats visa att medvetna avvikelser från dessa definitioner existerar. Vi har även presenterat vissa motiv till varför avvikelser sker.

Av de designkvaliteter vi undersökt har förståelse och modularitet genomgående under undersökningen genererat mest relevant data. Vi tror att dessa designkvaliteter ses som de mest tidskrävande – Intervjuperson A, som utvecklar i små projekt, ser det som tidskrävande att implementera dessa kvaliteter. Intervjuperson B, som utvecklar i ett större projekt, anser att efterarbetet blir större under drift om man inte implementerar dessa.

Således har vi, i vår undersökning, funnit en tydlig balansgång mellan implementation och drift när det gäller motivet bakom avvikelser från god design. Den vetenskapliga teorin för god design lämpar sig för större projekt, likt Intervjuperson Bs, då utvecklare ofta måste återvända till implementationer både under utveckling och under drift. Små projekt, likt Intervjuperson As, kräver inte lika många återvändon till implementationer som större projekt gör. Således blir de egenskaper, som förståelse och modularitet besitter, mindre relevanta för små projekt.

Dock räcker inte antalet återvändon som mått på om traditionell god design ska följas. Melissa et al. (2000) förklarade vikten av att komplexitet också definierar om ett projekt är litet eller stort. På samma sätt har vi valt att motivera om traditionell god design ska följas. Som exempel är Intervjuperson Bs större projekt väldigt komplext med många klasser och beroenden. Rimligtvis bör en hög modularitet och förståelse ha större påverkan på ett komplext projekt likt Intervjuperson Bs:

- En hög förståelse underlättar navigering och förståelse av funktionalitet, något som rimligtvis behövs mer i komplexa projekt.
- En hög modularitet underlättar testning, drift och utbyggnad. Komplexiteten i projekt försvårar test, drift och utbyggnad.

Vår slutats är sammanfattningsvis att medvetna avvikelser existerar för vårt undersökningsobjekt. Framför allt motiveras dessa avvikelser med sparade resurser och liten föränderlighet i slutprodukten. Dessutom påverkar utvecklarens personliga egenskaper och erfarenheter.

Avslutningsvis vill vi påpeka att det i uppsatsen varit komplicerat att formulera termen god design, då den teoretiska definitionen på god design i vår undersökning nödvändigtvis inte är god design för det lilla projektet. Vi har med vår undersökning visat att god objektorienterad design inte alltid är synonymt med god design.

6.1 Vidare forskning

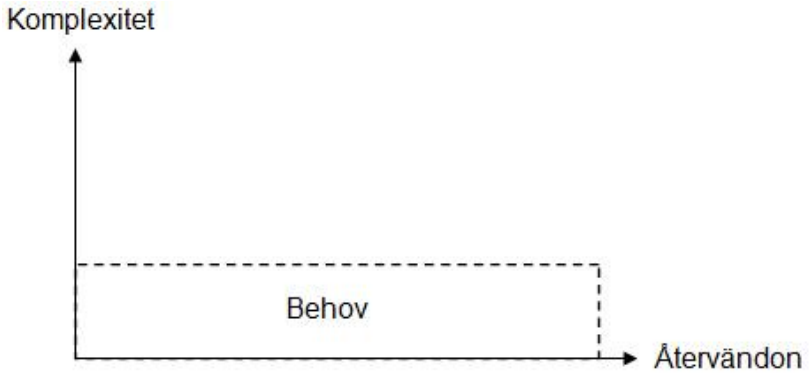
(1) Med vår undersökning ville vi påvisa eventuella medvetna avvikelser från god design, och sedan ta reda på vad motivet bakom dessa varit. Med vårt syfte och forskningsfråga har kravet på generaliserbarhet varit litet. Vidare forskning skulle i ett större sammanhang kunna visa vilka avvikelser som medvetet tas.

(2) Vi har inte mätt effekterna av identifierade medvetna avvikelser från god design. Vidare forskning skulle kunna visa om medvetna avvikelser i små projekt gynnar dessa. Resultatet av en sådan undersökning skulle kunna leda till att designlitteraturen måste göra skillnad mellan små och stora projekt, något som oftast inte görs i den objektorienterade litteraturen.

(3) Vidare forskning skulle, som vår slutsats förslår, kunna pröva om förhållandet mellan projektkomplexitet och antal återvändo motiverar, vad som objektorienterat anses vara, god design. Förhållandet vi föreslagit i slutsatsen är enligt följande:

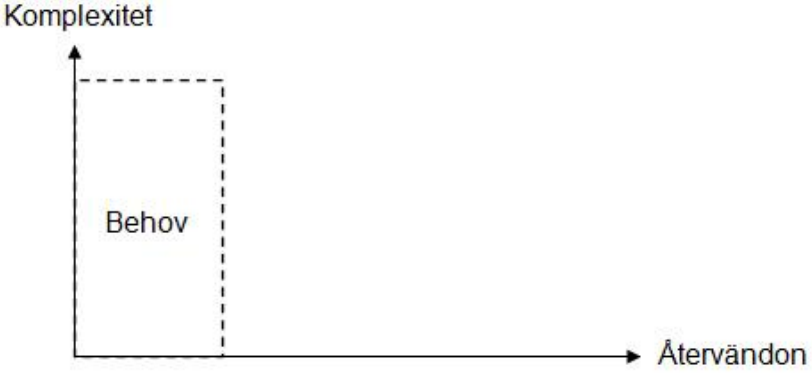
$$Behov\ av\ god\ objektorienterad\ design = Återvändon * Komplexitet$$

Enligt fallet i figur 6.1 är komplexiteten i ett projekt låg, medan utvecklarna ofta måste återvända till implementationen antingen under utveckling eller drift. God objektorienterad design besitter hög modularitet och hög förståelse, vilka skulle underlätta återvändona. Även om komplexiteten är låg så underlättas de många återvändona, vilka tillsammans resulterar i ett ej försumbart behov. Arean i figuren representerar behovet av god objektorienterad design:



Figur 6.1: Låg komplexitet & många återvändon

Enligt fallet i figur 6.2 är komplexiteten i ett projekt hög. Dock behöver utvecklarna sällan återvända till redan implementerad funktionalitet. God objektorienterad design besitter hög modularitet och hög förståelse, vilka skulle underlätta återvändona. Även om återvändona är få, så ökar komplexiteten i projektet resursåtgången i dessa, vilket resulterar i ett ej försumbart behov. Arean i figuren representerar behovet av god objektorienterad design:



Figur 6.2: Hög komplexitet & få återvändon

BILAGOR

Bilaga A – Intervjuguide 1

Diskussionsområde	Hjälpfrågor

Personlig erfarenhet	• Erfarenhet av utveckling?
	  ○ Antal år?
	  ○ Huvudspråk?
	  ○ Typ av projekt?

Projektutformning	• Antal projektarbetare? • Parallella projekt?

Skapandet av klasser	• Data/funktioner som definierar?
	• Resulterar i:
	  ○ Antal klasser?
	  ○ Sammanhållning?

Läsbarhet av kod	• Aktiv dokumentation av kod? • Talande klasser/metoder/attribut?
Förståelse	• Logiska kopplingar/beroende mellan objekt? • Verklighetsförankrade klasser/metoder/attribut? • Hur lätt är det för en nytillsatt utvecklare att förstå redan skapad kod?

Storlek	• Försöker ni hålla nere antalet rader kod?
	  ○ Bredd på arvshierarki?
	  ○ Multipla arv/interface?
	  ○ Koppling/beroende mellan objekt?
	  ○ Dokumentation i koden?
	  ○ Antal klasser?

Testning	• Hur utför ni testning av implementation?
	  ○ Steg för steg?
	  ○ Efter ett antal steg?
	• Brukar det uppstå några problem under testning?
	  ○ Multipla arv?
	  ○ Kopplingar/beroende mellan objekt?
	  ○ Inkapsling av privat data?
	  ○ Antalet klasser?
	  ○ Sammanhållning?
	• Hur viktig är förståelse för testning?
	  ○ Koddokumentation?
	  ○ Talande klasser/metoder/attribut?

Erfarenhet av drift	• Ansvarar ni för någon form av drift? • Bedrivit någon form av drift tidigare?

Viktigt med drift	• Vilka problem brukar uppstå vid drift?
	  ○ Förståelse?
	  ▪ Logiska kopplingar/beroende mellan objekt?
	  ▪ Dokumentation i kod?
	  ▪ Sammanhållning?
	  ▪ Talande klasser/metoder/attribut?
	  ○ Modularitet?
	  ▪ Multipla arv?
	  ▪ Utbytbarhet av arv?
	  ▪ Kopplingar/beroende mellan klasser?
	  ▪ Inkapsling av privat data?
	  ▪ Antalet klasser?
	  ▪ Sammanhållning?
	  ○ Säkerhet?
	  ▪ Djup av arvshierarki?

Bilaga B – Intervjuguide 2A

Identifierade designegenskaper

Implementation

- Funktionalitet
    - Varför skapa nya instanser av kfm flera gånger?
    - Varför bara en klass kfm som gör allt? (God class)
        - Varför inte bryta ut SQL-funktioner?
        - Varför inte bryta ut filrelaterade funktioner?
- Förståelse
    - Inte alla objekt är talande (till exempel ”button1”).
    - _variabelnamn används även på lokala variabler, men ibland inte.
    - Mer talande metodnamn. Vad är exempelvis XFR?
    - Vissa metoder väl dokumenterade, andra inte.
    - Många termer som verkar unika för företaget. Hur ska en utomstående utvecklare förstå dem?
- Storlek
    - SQL-metodernas namn är talande. Nödvändigt med dokumentation?
    - Varför endast en klass?
    - Återanvänder mycket kod. Varför inte bryta ut (DayStock-funktioner)?
        - Ny instansiering av kfm flera gånger.
        - File locations.
        - Ny streamreader istället för att nollställa den redan existerande.
    - Varför inte importera System.Threading då de används flera gånger?

Testning

- Hur används felhanteringen (1:or 0:or). Varför inte mer detaljer?
- Svårt att hitta var felen ligger om funktionalitet, SQL och filer ligger i samma klass?

Drift

- Förståelse
    - Inte alla objekt är talande (till exempel ”button1”).
    - _variabelnamn används även på lokala variabler, men ibland inte.
    - Mer talande funktionnamn. Vad är exempelvis XFR?
    - Vissa funktioner väl dokumenterade, andra inte.
    - Många termer som verkar unika för företaget. Hur ska en utomstående utvecklare förstå dem?
- Modularitet
    - Varför inte bryta ut SQL- och XRF-funktionalitet till nya klasser.
    - I property-funktioner görs mer än set/get.
    - Varför inte bryta ut kfm?
    - Vissa file locations är utbrutna, men används sällan. Vissa är inte ens utbrutna.

Generellt

- God sed
    - Dubbla return i funktioner används ibland.
    - Instansierar properties direkt i variabeldeklarationen istället för i konstruktorn.

Bilaga C – Intervjuguide 2B

Identifierade designval

Implementation

- Flera klasser utan instansvariabler.
- Flesta klasser saknar dokumentation.
- Sammanhållningen tycks vara väl konstruerad.
    - Väldigt breda arvshierarkier.
- MVC?
- Importerar sällan framework-klasser.
- BusinessLayerBase innehåller den mesta logiken.
    - Använder alla klasser all funktionalitet?
    - Bryta ut BusinessLayerBase?
- Hämtar en stor mängd data från databasen på en gång.
- Validering av personnummer på tre olika ställen.
- Brutit ut ASP-delar till kontroller.

Testning

- Väldigt breda arvshierarkier.
- MVC?.
- Property-metoder har mer funktionalitet än att bara tilldela och hämta.

Drift

- Hårdkodade fullständiga länkar.
- Hårdkodade databaskopplingar.
- Talande formkontroller.
- MVC?
- Klasser är väl utbrutna. Använder ofta partial klasser för att dela upp metoder från till exempel private-variabler och enums etcetera.
- Regions används flitigt i koden för att stycka upp och förenkla navigeringen.
- Property-metoder har mer funktionalitet än att bara tilldela och hämta.
- Viss information, så som hårdkodad laginformation bör brytas ut.
- Indelning i moduler.
- SQL-anrop från code-behind till ASP-sidor.
- Felhantering?
    - SQL-anrop utan felhantering.
- Bryter ut date format exempelvis.
- Vissa inställningar är utbrutna till en configurationsfil.

Bilaga D – Intervju 1: Inledande intervju med Intervjuperson A

[Fråga 1] Uppgiftssökare: *Har du erfarenhet av att jobba på små respektiva stora projekt?*

Informant: Alltså jag vet inte, men håller man på med utveckling på stora företag så har de ju oftast riktlinjer och jobbar man i grupp så har man också riktlinjer för hur det ska fungera och då följer man väl dem för att alla ser ens kod, än om man sitter i en hörna och jobbar. Om man jobbar själv eller på ett mindre företag där man inte har de här riktlinjerna så blir det så klart annorlunda av olika skäl.

Men jag vet inte.. det är ju kanske inte så konstigt. Så är det väl alltid liksom om man sitter och gör något hemma så tänker man kanske inte på det sättet. Det enda som jag tycker intressant är.. Eller egentligen om jag skulle vända på det, så det som är intressant är.. är det negativt för kunden i slutändan om det är internt eller externt är väl egentligen det som jag skulle tycka är intressant. Är det negativt.. jaha.. hur ska man göra åt det då kanske?

[F2] U: *Tanken är ju att det är vårt nästa steg om vi hinner, men vi vill först visa på att det faktiskt finns avvikelser.*

I: Men det är ju rätt så naturligt att.. Jag tror ju det finns flera faktorer.. För det första, de som varit med på stenåldern då, som jag nästan har varit, som arbetat med utveckling innan objektorientering fanns. Till exempel om man tittar på klassisk C. Det är ju inte riktigt objektorienterat på det sättet. De har ju lärt sig att utveckla på ett visst sätt, och när det då helt plötsligt kommer en ny företeelse så är man.. en människa är ju ett vanedjur. Har man lärt sig någonting så vill man ju gärna fortsätta på det spåret.

Det är en sak som jag sett när jag arbetat under de här åren. Man ser att de som har arbetat med RPG eller någon annan utveckling tidigt som nu helt körts förbi av den här objektorienterade världen, de skriver väl kod lite annorlunda. För det första har de kanske inte någon formell utbildning inom det, utan de har kanske lärt sig själv, för de har ju ändå någon utvecklingsbakgrund i botten då. Och då gör de väl lite som de själva tycker.

Det är väl kanske den första biten jag kommer att tänka på.. Att man är just vanedjur.. att man kommer från den.. den biten. Man kommer från den bakgrunden, alltså att man utvecklat i andra språk. Men sen tror jag.. alltså sen är det nästan det ni försöker bevisa nästan är en självklarhet att folk försöker göra det mer effektivt.. göra det snabbare och kanske inte alltid följer reglerna.

[F3] U: *Mm, har du något exempel kanske?*

I: Ja.. Eller kanske inte något bra exempel så liksom men.. Ofta är det ju så att om man jobbar så som konsult, kommer en kund och vill ha något väldigt snabbt. Alltså, det absolut vanligaste felet som händer är att kunden kommer och säger: ”Du, jag vill ha en sådan här pop-up”.. Och så frågar man då lite om requirements och kravspecifikation, och det har ju såklart kunden. ”Jag ska bara ha det här, och det ska bara stå en enda sak i den där rutan. Det ska bara stå ABC. Hello world ska det stå”. Jaha, tänker du. Då slänger jag bara ihop något, för det är allt han vill ha.

Så går det två veckor så kommer kunden tillbaka och säger: ”Du, det här är ju skitbra. Skulle jag kunna få en knapp under här? Och när jag klickar på den så vill jag att den ska göra så och så och så”. Och då har du kanske gjort projektet.. När du startade det så kanske du bara slängde ihop någonting. För det var det enda kunden ville ha. Och helt plötsligt fastnar du i ett skäl. Då måste jag fixa det. Är du då under tidspress kanske du smäller ihop den knappen också, du bara trycker dit en knapp som gör någonting.

Och sen när du är färdig med det så kanske det går ytterligare två tre veckor och då kommer kunden igen och säger: ”Du, det här är fantastiskt, men nu vill jag ha detta och detta och detta”. Och det är väl oftast så det börjar liksom. Alltså, jag tror det är sällan man sätter sig ner, eller det händer såklart, att man sätter sig ner och har en väldigt detaljerad kravspecifikation.. Man vet vilka deliverables man har. Man vet vad man ska levererar och så vidare och så vidare. Så är det ju.

Många projekt börjar med att man gör något litet och sen växer och växer och växer, för folk är intresserade och vill ha mer. Och då blir det ju naturligt att man slänger ihop kod. Alltså, om man jobbat länge.. Jag kommer ihåg när jag jobbade i USA, så gjorde vi ett sådant här helpdesk-system.. på nätet. Det var exakt likadant. ”Du, vill bara kunna se alla ärenden. Det är det enda vi vill göra”.

Och då tänker man; jaha.. på den tiden det fanns ASP.. det är ju ett scriptbaserat språk ju. Amen då slänger jag bara ihop en sådan sida och så gör jag bara en koppling till deras system X. Och så visar jag bara. Ingen säkerhet eller någonting. Man ska bara kunna fylla i sitt ärendenummer och så får man se ärendet. Och så fort man lagt på det så säger folk: ”Du, det var ganska smart. Kan jag få se alla mina ärenden? Alltså jag skriver mitt namn så vill jag se alla mina ärenden”. Jaha, ja men det går väl.

Då ändrar jag den som frågar efter ditt ID så skickar jag tillbaka alla ärenden. Så börjar folk säga: ”Du, det där är jättebra. Skulle jag kunna klicka på mitt ärende och se om det är kopplat till några andra ärenden?”. Och då helt plötsligt är man igång. Då har man börjat med en ASP-sida vid sidan om som folk tyckte var kul.. och helt plötsligt är det hundratals användare som tycker det här var bra. Och koden då har blivit ett hopkok liksom. Men jag gjorde bara något för att folk ville ha det.

[F4] U: *Tror du det är för lite designförberedelser i just små projekt? Jag tänker liksom på UML-diagram och sådana här grejer.. Att man funderat och analyserat?*

I: Absolut. Ofta blir det ju så att man inte har den tiden på sig att göra det. Och det är väl kanske det som är det intressanta med det ni gör här. Att diskutera om det är bra eller dåligt. Felet är.. Om man går tillbaka till det här exemplet med pop-upen. Om man bara ska göra en pop-up med en knapp som bara ska göra en jätteliten sak, och kanske.. och kodantalet kanske är 50 rader kod. Då sitter man ju inte och gör en UML. Man sitter ju inte och skriver massvis med dokumentation runt det här man har tänkt, utan man slänger ihop det.

Sen när man ska bygga på det projektet. När i den här kurvan ska man liksom börja med dokumentation? När känner man att projektet är så stort att nu måste vi börja dokumentera? Det känner man aldrig, för man bara bygger bygger bygger bygger bygger bygger. Man kan jämföra det med folk som hemmarenoverar. Finns ingen ritning kanske, utan vi ska sätta upp en vägg här. Jättebra och så gör man det. Och sen, ”Kan vi inte göra en snedvägg här och en dörr där?”. Men sen när man har bott i det huset i 20 år och gjort alla de här, så kommer någon annan och frågar: ”Du, har du någon ritning på hur huset ser ut?”. ”Nä, jag har bara ritningen som det såg ut när jag började”, säger du.

Lite så är det kanske. Det blir ofta att man börjar på ett sätt för att göra något snabbt.. och sen blir det kardinalfel. Och när man kommer den här långa resan som går i ett projekt, så blir det att det blir ett hopkok. För det är aldrig någon som tar tag i det. Eller vissa delar dokumenterar man, men inte vissa andra. Sen.. I slutändan tror jag det är mycket självdisciplin. Att utveckla för mig är ett hantverk, som vilket hantverk som helst. Så det är som att vara snickare eller plattsättare eller takläggare. Och om du då gör något dag ut och dag i in så lär du dig ditt sätt att göra det. Och det är kanske bra, det är kanske dåligt eller okej liksom. Men så blir det för att du lär dig att så och så ska man göra.

Än en gång.. människan är ett vane.. man vänjer sig vid vissa saker. Så är det alltid. Jag håller på med ett projekt nu till exempel, där jag ska göra en login-funktion då. Jag tittar bara på förra projektet jag gjorde.. och så blir det ju ’copy/paste’.. och då kanske man lägger till massa funktioner. Funktioner som inte är användbara, men om jag ska göra ett projekt och någon säger till mig.. så sätter jag mig ju inte ner med ett blankt papper. Utan då återanvänder man erfarenhet och/eller kod man har. Och är den koden då inte dokumenterad så blir ju inte nästa bit dokumenterad heller liksom. Så det där går ju lite hand i hand.

[F5] U: *Brukar det se bra ut med dokumentation ute i projekt. Eller är det något man brukar slarva med?*

I: Det slarvas alltid. Och grundskälet är ju att nästan alla som utvecklar har ju någon form av teknisk bakgrund. Alltså, de är ju inte designers eller grafiker eller författare. Nej, de är utvecklare och ni vet ju själva när man gör någonting. Det stora nöjet med att utveckla är ju att lösa något problem. Alltså, att få det på plats. Att göra något med så lite kod som möjligt, gärna med en enda rad om det går, och så är den så här lång, men det känns jättebra när man tittar på den. Wow, bara en enda rad gjorde jag. Och det klassiska; äh, jag skriver en kommentar sen. Så glömmer man det.

Men det är ju det som driver oss som är tekniskt intresserade. Det är ju inte själva lösningen. Sen att skriva en hel manual på 100 sidor är kanske något vi tycker är tråkigt då. Tycker man något är tråkigt, att sätta på diskmaskinen hemma, så skjuter man gärna på det. Och så kommer det massa saker emellan och så glömmer man det. Så, jag tror att det är jättevanligt.

De enda gångerna man har det väldokumenterat är om man har en chef som är väldigt strikt. Alltså, som kräver det. Och har man inte det här kravet på sig så tror jag att ytterst få gör det av egen fri vilja. Men egentligen är ju

den grundläggande frågan, hur viktigt är det att göra all den här dokumentationen? Och hur viktigt är det att göra det här diagrammet och att ta ett steg tillbaka och liksom försöka utvärdera det igen? Det är väl nästan det som är det intressanta.

[F6] U: *Är det ofta man.. Du som jobbat i små projekt. Att man får gå tillbaka till projekten efter att man levererat?*

I: Alltid. Det blir alltid så att du utvecklar någonting. Och sen ger du det till någon. Och så börjar de att använda det. Och vi är ju så som människor. Att vi direkt.. Jag skulle nog kunnat göra så och så. ”Alltså, hade jag bara haft den här knappen här så hade allt gått 100 gånger snabbare”. Och då går man tillbaka till utvecklaren då. Så får man gräva fram det och lägga till en knapp då. Så går det ytterligare några månader så kommer folk på ytterligare idéer.

Det enda som egentligen styr det i slutändan är ju pengar. Hur mycket pengar är företagen, både internt och externt, villiga att lägga på de här projekten? Det är egentligen bara det som styr. Så det sker jätteofta. Det största felet är egentligen att.. när det går mellan två utvecklare. Så även om jag går tillbaka till kod som jag skrivet för jättelänge sedan.. Vi pratar kanske sex, sju, åtta, tio år sedan. Så i och med att jag gjort det i min egen stil då, så vet jag kanske varför jag gjorde så. Kanske inte exakt varför jag gjorde så och så, men någonstans där i bakhuvudet kanske man kommer ihåg att jag gjorde nog det för det och det och det.

Men när jag får kod ifrån ett annat projekt. Jättesvårt, jättesvårt. Ignen aning hur den personen har tänkt. Eller.. Varför gör de så? Ibland tänker man: ”Åh, han måste vara helt korkad”. Bara för han gjort så här. Men det kanske fanns ett jättebra skäl när han gjorde det. Eller så kanske han inte visste att man kunde göra det så. Själv när jag går tillbaka så kan jag tänka: ”Men varför gjorde jag så? Det här var inte så smart liksom”.

Det är ju självklart viktigt att dokumentera. Men jag tror ibland.. Det är som allt annat i livet. Att hitta en balans. Egentligen ska man bara dokumentera det som är avvikande. Om du har en klass som hämtar ett värde från en databas. Behöver du skriva en kommentar: ”Detta är en klass som hämtar x från databasen”? Nej det måste man väl inte. Vilken utvecklare som helst tittar på koden och ser att där finns en klass som hämtar detta värdet från en databas. Måste man skriva en kommentar där då? Måste jag skriva att jag skriver in en string och får tillbaka en string? Det ser ju jag i den tabellen. Då är ju frågan; är det intressant?

Men om jag har en metod som beräknar någonting, gör en jättekonstig beräkning. Där är det kanske intressant att skriva upp varför man tänkte som man gjorde. ”Varför gjorde jag så? Ah, det var så och så och så”. Jag såg ju ett diagram med kommentarer och pilar hit och dit. Det blir nästan svårare att tyda då. Det här projektet måste vara en miljard rader kod liksom. För allt detta här..

Och controller. Som man ska göra ju. Men i sådana projekt tappar man liksom lite av innebörden. Att behöva gå via den hela tiden. Och jag vet ju att de som är sådana här ’freakar’, de vill att det ska vara så. Och hade jag gjort ett google.com så hade det kanske varit jättesmidigt. Men om man gör en sida som ska användas av hundra personer, varför ska jag göra det då?

[F7] U: *Så, struntar du ibland att använda mönsterlösningar? Just för att hålla det enkelt?*

I: Absolut absolut absolut. Och det är ju samma sak.. Det finns ju massa på Internet om SCRUM och så vidare. Det har ni säkert tittat på ju. Sen finns det ju också.. Det finns ju också många projekt som är ’throw-away’. Man gör ett projekt och sen ska man inte använda det mer. Till exempel om man ska göra en engångs-import av någonting. Men den är ändå väldigt komplex. Man måste ändå skriva det här verktyget för att hämta ut data. Sen måste man hålla på och konvertera och mecka. Troligtvis kommer man inte återanvända den då.

Måste man då göra den enligt konstens alla regler? Jag ska ju bara göra det enda gång. Varför då dokumentera? Det är ju att skriva och fundera. Jag bara sätter mig ner och så försöker jag göra det. Och lösa det problemet just då. Jag tror.. Och alla jag känner. Sitter man själv eller inte är strikt styrd av en stor organisation så blir det att man gör lite hur man vill. Alltså det bli genvägar. Och sen.. Ibland är det bra, ibland är det dåligt, ibland får man gå tillbaka till den där genvägen så kommer man på..

Jag kan faktiskt ta ett exempel. Jag har gjort en pop-up till företag X, och där stoppar man in ett personnummer och så får man tillbaka information från deras system. Adress och telefonnummer och så vidare så att man ska kunna ringa den här personen. Och när jag träffade den här killen för första gången sa han: ”Du, jag vill bara ha

en enkel pop-up så. Man ska stoppa in personnummer, så ska man bara få ut namn, adress, telefonnummer och några grejer liksom". Jaha, tänkte jag då. Inget annat? "Nej, det ska bara vara så".

Ja, men då gör jag en snabb.. Jag gör en Windows-lösning, tänkte jag. Det är ju det absolut enklaste. Och så satte jag upp ett litet.NET.. Allting är ju.NET, för det är det enda jag sysslar med. Jätteenkelt. Ni vet hur det ser ut. Man får ett form man bara drar dit. Där ska man ha det och så vidare. Och så visade jag det för kunden. "Åh, jätte jättebra". Så gick det bara någon dag sen. "Du, kanon. Jag skulle ville hämta information från två andra system och så skulle jag vilja visa den här". Jaha, nä det visste jag inte om jag ville göra.

Windows forms är ju inte likadant som webben. Nu hade jag ju redan installerat den här på sju olika klienter. Aja, då fick jag gå tillbaka och lägga till det. Installera om den på alla de här sju personerna då. Och efter bara någon ytterligare vecka så kom han: "Du, det här är ju skitbra. Nu vill vi verkligen ha de här bitarna och en uträkning här baserad på hur mycket de är skyldiga oss". Och då hade jag redan kommit så långt.. Då hade jag redan spenderat en vecka.. nej kanske inte vecka, men ett par dagar i alla fall.

Och då kom jag fram till att det här kommer ju inte fungera liksom. Jag kan ju inte springa runt varje dag och byta ut de här klienterna. Då fick man göra så här istället. Då fick jag göra om hela pop-upen och göra den som en stor ruta och i rutan så laddas en sida. Och då får jag ju en slags 'clienteer-applikation'. De har ju bara en liten pop-up på sin, men den ändrar ju inte sig. Och så matar in ett personnummer, som visas på webbsidan och så visas det som en intern browser.

Helt plötsligt kunde jag ju lägga till funktioner när som helst. De tror ju att det är en klient, för de ser ju inte det liksom. Men egentligen laddas den från en webbserver som de har hos sig. Och det är väl typ något sådant som man slänger ihop.. Nu var det ju tur att jag stannade så tidigt, för jag hade ju kunnat fortsätta med den här 'Windows-appen', som många gör. Ofta fortsätta utveckla. Sen är du ju inne i en gränd till sist. Det blir så enormt mycket jobb att underhålla och uppdatera det här.

Slutsatsen blir väl att man ska ta sig en funderare på hur stort projektet blir. Och egentligen ska man väl göra som så att man ska fundera hur stort det kan bli och hur mycket tid man ska lägga på dokumentation och de bitarna. Jag tycker; ska det vara ett litet projekt så ska det vara lite dokumentation. Inte i förhållande till projektet, för det blir ju mindre. Men generellt. Ska du göra ett jättestort projekt så ska du ha mycket dokumentation. Ska du göra ett mellanstort projekt så ska du lägga dokumentationen där emellan.

Jag tror inte att en lösning passar alla. Man kan inte säga att; varje gång du gör ett projekt så ska du göra alla de här tio grejerna, som de säkert sa till er i skolan. Det kommer jag också ihåg. Så är ju inte livet. Det är ju inte så lätt att varje gång du gör något så ska du göra detta. Utan jag tror att ju mer erfaren man blir som utvecklare ju lättare kan man se hur stort systemet blir och hur mycket tid man måste lägga på det. Och det är kanske de bitarna som ska styra, snarare än de bitarna som står i skolboken.

[F8] U: *Vi har rört vid ämnet, men just drift.. Jag tänker då dels på uppdatering av programvara, men även att byta ut delar.. Om det skulle vara lättare om man på något sätt strukturerade upp det ordentligt från början?*

I: Både jag och nej för att.. Om du ska byta en del så är den ofta kopplad till massa andra funktioner även om den är väldigt väldefinierad och har bra interface. Får kanske svara lite ja. Om du har definierat allting extremt väl så vet du exakt vad den komponenten gör och vilka andra delar som använder den komponenten och hur de använder den. Då borde det ju vara lättare alltså. Men, jag har liksom inget sådant exempel där jag kan säga att vi definierade det jättebra och att det blev mycket mycket bättre.

[F9] U: *Men när du sitter och skriver din kod, då är det inget du har i baktanke att "Hur kommer det här fungera efter produktion om de vill att jag ska byta ut detta"?*

I: Jo det har man ju. Men det är ju kanske inte på den.. Nu pratar vi ju dokumentation, det är ju då annat från design. Men en sak med design är ju att såklart att tänka lite: "Hur ska det här underhållas?". Jo det är en person som sitter där och där. Vill jag kompilera om projektet varje gång jag ska lägga till en liten grej här? Det vill ju jag inte. Så hur kan jag göra? Jag kanske lägger en xml-fil någonstans. Då läser den ju xml-filen, så kan jag bara ändra i den och lägga till någonting. Då vet jag ju att jag inte behöver kompilera om programmet, så.. Men det är nog mer design än dokumentation. Att man tänker efter före. Det gör man ju. Lite hur det tekniskt ska fungera.

[F10] U: *Jag tänkte på det här med storlek. Du sa att det var att man försöker hålla ner koden så mycket som möjligt. Att det var lite av en personlig utmaning. Men det säger ju lite emot det här med att när du tar 'copy/paste' och det ligger kvar extra funktioner?*

I: Ja, med det är ju ofta när du gör något nytt så att säga. Så du vill hålla nere.. Det är ju nästan varje dag som man får någon ny idé eller lösning som man försöker implementera. Gör man 'copy/paste' så får man ju hoppas att man skrev bra kod förra gången. Man är ju aldrig perfekt, man försöker hela tiden utveckla sig själv till det bättre.

Men jag tror att i objektorienterad programmering så återanvänder man objekt naturligt så att säga. Det blir inte 'copy/paste' på det sättet. Man ärver ju från en annan klass så att säga, kanske lägger till några nya metoder och så. Men när det gäller dokumentation så vet ju inte.. Det är ju två olika saker och jag vet inte riktigt om de hör ihop så att säga. Jag vet inte riktigt var du vill komma?

[F11] U: *Jo vår uppsats handlar ju mycket om design. Alltså "Hur implementeringen" påverkas och så vidare. Och vad jag då vill komma till är "Hur viktigt är det att hålla kodstorleken nere i ett projekt?"*

I: Egentligen, som jag sa innan, så tror jag det mer är en personlig utmaning. Det finns egentligen inget, vad jag vet i alla fall, som kompileringsmässigt skulle göra så att det tar längre tid. Det är så att man vill att det ska vara så 'tight' som möjligt, det är en sådan grej liksom. Man skriver inte IF IF IF, utan man försöker trycka ihop det så mycket som möjligt, som man kan.

Men jag vet egentligen inte, om jag ska vara ärlig, att det bli bättre sen. Det blir ju oftast komplicerat sen. Du trycker ju ihop något som du tycker är självklart, men som någon annan inte förstår. Hade man brutit ihop det i delar så får du ju en bättre överblick. Jag tror inte det har någon påverkan så att säga.

[F12] U: *Har du några andra problem som du kan se? Som vi inte redan har nämnt då. Som kan uppstå om man slarvar i designen i mindre projekt.*

I: Ja det klassiska är ju det här att, som vi nämnde innan, man tänker inte på att det här ska uppdateras. Man stoppar in en massa saker i koden, 'connect strings', ja vad det nu än är. Man sätter ihop det, slänger ut det, och sen är det färdigt så att säga. Och sen så när det är klart, så ska man plötsligt flytta allt det här till en annan server. Oh shit! Tänker jag ju då. Nu ska jag in där och leta upp alla 'connect strings' jag hade överallt.

Det är ju samma som om man titta på nätverk. Något klassiskt är ju att man lägger in ip-adressen. Det är ju inte så smart, för sen kommer kanske nätverkskillarna och säger "Du, vi måste flytta det här till en annan ip-adress, men det är samma dns-namn så du behöver inte oroa dig". Ja det gör ju ingen skillnad, för nu skrev jag ju in ip-adressen, man gjorde det ju så snabbt liksom.

Man tänker inte det här extra steget. Men tror att den största skillnaden är att man inte behöver sätta sig ner i ett mörkt rum i fyra timmar och fundera på design om man ska starta ett nytt projekt. Men man ska ta ett steg tillbaka och tänka lite när man börjar. Men sen är det ju mycket erfarenhet som spelar in, om till exempel hur man tror det ska börja implementeras och användas.

Jag tror det är mycket sådana klassiska fel man gör med design. Man tänker inte riktigt efter. Sen en annan klassisk sak man gör, om vi tittar lite här på webbtjänster, att man bryter inte ut det tillräckligt utan man återuppfinner hjulet genom "copy/paste" om och om igen. Man hade kanske kunnat byggt en klass man hade kunnat återanvända. Kanske en webbtjänst egentligen.

Man tänker ju på att nu ska jag ju göra 50 applikationer på webben, och alla ska ha ett login-interface. Egentligen hade jag kunnat byggt en login-modul och satt som en webbtjänst. Sen skickar jag bara applikationsnamn, användarnamn och lösenord till den. Sen verifierar den och skickar tillbaka det. Varje projekt jag hade sen hade ju bara kunnat återanvände det, om och om igen.

Men när man börjar med första projektet så tänker man ju inte på det att nu ska jag nog göra 50 projekt, så jag ska nog göra allt detta jobbet för att göra den. Sen efter 20 projekt så tänker man, usch nu måste jag tillbaka och implementera den igen. Och då skiter man i det. Man har liksom låst in sig i en hörna. Men än en gång, är det bra eller dåligt? Varje projekt är ju för sig. Jag har ju inte en komponent som kan fallera. Alla projekt är ju för sig själva.

[F13] U: *Och just det här med vad du sa om klasser är väldigt strikt inom projektorientering. Det ska inte finnas klasser som bara har variabler, de ska ha verklighetsförankring, och så vidare.*

I: Alltså, de saker jag brukar följa är det sista. Att de ska ha någon form av verklighetsförankring. Ett objekt är ju ett objekt. Som en stol, ett bord, en dörr, eller vad som helst. En stol har ju då 'properties'. Den har ett säte, den har armar och så vidare. Så den biten kan jag tycka är ganska okej, och man försöker ju hålla det, men det är inte alltid så lätt.

En kontakt eller en kund är ju ganska enkelt. Men ibland kommer ju saker som, t.ex en order. Den har liksom olika aspekter. Den har en 'header', den har en 'details'. Hur ska jag liksom få ihop allt det här? Då blir det liksom inte så enkelt. Är order ett objekt? Det är det ju inte egentligen. Det är flera objekt så jag vet inte riktigt.

Än en gång tror jag inte det är så extremt det där med objektorienterat. Det är inte alltid rätt. Vem säger att jag inte bara kan göra en klass? Vem säger att det är fel? Blir det sämre när jag kompilerar? Det finns ju ingen som säger det. Det är ju då någon som har kommit fram till en massa regler, som egentligen är så kallade 'best-practices' vilket egentligen inte är regler, utan mer att "Vi uppmanar dig att göra så här".

Ni vet ju t.ex hur det är att ta körkort. Handgreppet tio-i-två. Det är ju inget som säger att det blir bättre. Det gör ingen skillnad om jag håller lite längre ner, men det är ändå så de säger. Att man måste hålla så, annars blir man underkänd. Och exakt så kan ni tänka på när ni håller på med utveckling. Någon säger "man måste göra så här!", men så är det egentligen inte. Man måste inte göra så, det gör ingen skillnad. Det är som när man haft körkortet i tio år så sitter man inte så med händerna.

Så är tanken tror jag, att det finns så många regler och 'best-practices', men sen när man börjar jobba med det så hittar man sin egen stil. Om man tar det här med körning igen. Alla har ju olika körstil. Men det betyder ju inte att den ena kör bättre än den andre, och samma gäller inom utveckling. Det enda man ska tänka på är viktiga klassiska fel som vi sade.

Allt som måste kunna ändras enkelt ska ju ut ur koden till xml-, ini-, eller config-filer. Man ska ju inte skriva samma kod om, om, om, och om igen. Det är ju helt vansinnigt. Då får man ju lägga det någon annanstans och hämta. Som en statisk metod eller funktion. Men hur man exakt designar en klass, eller ett objekt. Det är inte så strikt alltså. Om man tittar lite på det här med 'proporties' inne i en klass så kan man ju göra mycket där. Men de kanske säger att: "Nej så där ska du inte göra. Du ska bara hämta och lämna här". Men det tycker jag man kan göra. Jag tror det är som med allt annat här i livet, det handlar om suft förnuft.

[F14] U: *Vad vinner man på med allt det här som vi har diskuterat om?*

I: Det är svårt att sätta sig in i det här men säg att du hade gått ner och satt dig på Ericssons mobilutvecklingsavdelning. De säljer en telefon som används av miljontals, miljontals människor världen över då. Telefonen måste ju fungera. Det får inte finnas en sådan klassisk bugg som om man trycker in 1 och 3 samtidigt så kraschar den liksom.

Och ni sitter ju i ett sådant ENORMT komplex med hundratals utvecklare och ska göra den här lilla biten att om man trycker på 3, så ska det och det hända. Och så säger de till den här andra utvecklaren: "Du ska göra det här, och du det här, det här, här, här". Och sen ska jag då ta alla de här bitarna, den här koden då och trycka ihop det liksom. Och har då den ena utvecklaren suttit och gjort 70 klasser för att allting var ett objekt tyckte han, medans den andre utvecklaren: "Nej alltså. Bord och stolar. Det är ett objekt! Vi kallar det bord-stolar".

Sen när man då ska trycka ihop allt det här så tänker man: "Det här var ju inte lätt". Det är ju när man arbetar i stora projekt och är många som ska samsas om och då gäller det ju att ha regler för annars blir det ju jättekonstigt. Men om du sitter själv, eller om man kanske är två personer. Man lär ju sig hur den andra funkar liksom, "han skriver så och hon skriver så, jaja". Så att det är ju därför man måste ha reglerna, men då enbart för stora projekt. Det finns ju ingen annan praktisk funktion, förutom då att det är god sed att göra det.

[F15] U: *Så i små projekt vinner man i tid då eller?*

I: Ja jättemycket! Oj vad snabbt det går. Man slänger ju upp projektet på nolltid liksom. Sen är det kanske inte väldokumenterat, kanske inte optimerat. Men det gör ju ingen skillnad i slutändan. Än en gång att om du gör, jag kan tala om att man försöker optimera det. Det gör man ju undermedvetet. Man kallar ju inte på databasen 50.000 gånger per sida, det gör man ju inte liksom. Det känns ju helt irrelevant och tar då för långt tid för den

sidan att laddas. Utan man gör ju ett ’call’ till databasen och försöker hämta så mycket information som möjligt för att sedan sortera det på sidan. Det går ju snabbt när man har det i minnet på servern ju.

Jag tror faktiskt att man gör det undermedvetet. Man sätter ju sig inte ner och studerar det liksom. Nu ska jag tillverka denna funktionen, nu ska jag göra på alla de här olika sätten. Utan jag ska ju trycka ihop den här, och den här ’stored proceduren’ som jag gör. Den ska ju vara så optimerad som det någonsin går. För då är man ju på jättestora projekt. Då är man ju Google liksom som håller på med sökmotorindexering och ska indexera en miljard sidor varje kväll. Det måste ju vara väldigt bra gjort då.

Men om du göra en applikation då som de flesta av oss gör. Som används av mellan 100 till 1000 användare. Då kommer ju det inte påverka någonting för att datorerna idag är så snabba och fungerar så bra. Det är ju också en sådan klassisk sak att alla de reglerna kommer ju ifrån den gamla DOS-tiden.

Jag vet inte om ni kommer ihåg det, men man hade ju 640k och man kunde fixa ett sådant ’extended memory’ och sedan ladda upp det i detta ’extended memory’. Ja det fanns massvis med grejer och det är ju något man ser än idag. När ni går in, och ska sätta en variabel. Om ni tittar i en databas, till exempel SQL så ska man sätta till ’char’ då. Vilket är tecken då va, och sedan hur många tecken. 8, eller 10, eller 12 då va.

På gamla system ska ni se att det är supertight alltså. Och med ett personnummer då som är 10 tecken, så är det en CHAR[10]. Du får inte sätta 11 tecken för då kraschar det direkt. För förr i tiden var det ju dyrt med ’disc space’. Nu om jag hade gjort något sådant hade jag slängt in en VARCHAR[150], trots att jag vet kanske att det aldrig kommer bli mer än 10, men det kostar liksom ingenting.

Det är ju också en annan grej med design att idag är allt med datorer så snabbt, så det behovet som när med någon av de här språken, C till exempel, det är ju en mardröm. Pointers och allt att tänka på. Vilket meck! Förr i tiden var det viktigt ju. Var i minnet är det? Vilken adress ligger detta på? Då måste jag dit där, hämta det, för att sedan kopiera det någon annanstans för att gör något annat.

Så tänker man ju än idag, allting är ju objekt. De språken vi använder, Java och.Net är ju helt objektorienterat. ’Copy-file’ till exempel i C. Då pratar vi mycket kod för att bara kopiera en fil. Och så går du in i.Net och tar File, ’File-copy’ och allt kommer här och så fixar den allting till dig. Det är ju nog mycket det att de har ändrat sig så dramatiskt de senaste 20 åren.

Hur det var förr i tiden när man tittar på RPG, och C.. och Assembler ska vi inte prata om liksom! Där var det extremt viktigt det där med att hålla reda på ’memory-adresser’ och allt sånt där. Så är det inte längre idag ju. Så mycket av de som skrev de här ’best-practices’ har varit med och jobbat med, från början med det här. Och då blir det nog att de tar de erfarenheter som de hade från den storhetstid de hade. På så sätt har det blivit väldigt mycket regler som egentligen inte har någon riktig förankring. Man behöver inte göra så.

Samma med databaser. Idag så.. när jag gick i skolan. Då pratar vi om 90-, början på 90-talet. Så kommer jag ihåg den professorn vi hade där, systemteknikern, jag kommer inte ihåg vad han var riktigt. Vi kom in en dag där, när jag var 20 år eller vad det nu var, så sa han ”du vi har fått nya datorer” sa han. Vi ska hålla på med Assemblerprogrammering, för det höll vi på med då. ”Ni kommer inte tror det här” sa han. ”De här har 80mb hårddisk” sa han, och i Assemblerprogrammering så pratar vi om kilobytes alltså. ”Wow, 80mb” sa jag, ”Det är det största jag hört i hela mitt liv” sa jag.

Vi kom ju från de här 5 ¼ ” disketterna och liksom då var man tvungen att tänka så liksom. Jag måste vara väldigt noga med allt jag gör, måste vara noggrann. Jag har inte råd att kasta ut grejer. Tänk idag! En vanlig hårddisk har ju 500gb. En vanlig disk liksom. Du kan ju skapa en miljon skräpfiler. Det kommer inte märkas.

Så jag tror i grund och botten att det som ligger, det ni läser idag är skrivet av människor som utvecklade och lärde sig under en helt annan tid än än den vi lever i idag. Och jag tror att om 10 år kommer det att se annorlunda ut. Då kommer den yngre generationen att ha tagit över och då är kanske de, i den positionen att de kan skriva böcker eller hålla handledning om utveckling. Då kommer deras erfarenheter att ta över. De kommer liksom säga att ”Nä, så här ska man inte göra. Det är bara larv. Så här gör vi.”

[F16] U: *Så det har liksom gått över från att vara ett storlekshinder till ett tidshinder?*

I: Ja det tror jag absolut! Vi lever väldigt stressat idag och tempot är.. Jag pratade med min bror häromdagen, han, han är 10 år äldre än mig och när han började jobba, då fanns ju inte mobiltelefoner. Och det tycker man

låter helt overkligt. Då fanns inte mobiltelefoner, och man hade nästan precis fått fax. Han hade alltså då inte ens fax, och han jobbade då som säljer då va. Då ringde då kunden och sa ”Du jag vill ha produkt XYZ och en offert på det”.

Då satt min bror vid skrivmaskinen och skrev ihop offerten, vek ihop den och la i ett kuvert, skickade, och två dagar senare ringde kunden ”Du det var fel på offerten. Du får rätta lite på priset”. Då skrev han ut en ny offert till kunden, skickade den, och två dagar senare kom det tillbaka. Han höll på så och det kunde gå en hel vecka innan de ens hade kommit överens om priset. Idag ringer man bara ett samtal, skickar ett e-mail och så är det klart. Det sker ju på nolltid.

Tempot på samhället har ju ökat dramatiskt och då måste man ju själv hänga med. Man ser ju hur snabbt det har gått till 140.000 ’appar’ på AppStore. Och hur länge har det funnits? Ett år? Inte ens det? Kanske ett år? Då har de alltså utvecklat 140.000 appar! Det måste sitta någon som jobbar som en galning! De har utvecklat spel och allt vad det är. Då har de ändå lärt sig en ny teknik ofta, för att de kanske inte är apple-utvecklare.

Så att tempot, tiden, jag tror det är precis som du säger. Jag tror det är en bra slutplädering är att tiden kommer att.. Allting måste bli mer effektivt och då har man inte tid med traditionell utveckling som har sina rötter i 80-talet skulle jag säga. Där de utvecklade på ett helt annat sätt. De hade gott om tid, var noggranna, hade ont om minne, ont om disk. Idag har de gott om disk, gott om minne, bra linor.

När jag tänker på TCP/IP, hur det är uppbyggt, hur det fungerar, grund och botten. Titta på hur det är byggt, paketstorlekarna som skickas över nätet då, hur stora varje paket är då de kommer fram, hur de kommunicerar. Allting var byggt på jättesmå linor. Vi pratar liksom, modem 14.4k liksom. Det var det gjort för. Idag sitter man ju på 100mbit där hemma. Det är ju samma sak.

Det kommer ju att ändras, så det är kanske det ni ska använda som slutplädering. Att det är det som.. Det är bara som jag teoriserar, men jag tror det ligger mycket i det att reglerna är skrivna för så länge sedan att människor som hade en helt annan inställning och helt andra behov och krav på sig. Då låter ju inte som så länge sedan 20 år. Men i vår värld är ju 20 år.. Tänk 20 år framåt. Då sitter vi inte med sådana här mobiler. Då har vi en magisk grej som vi bara touchar som sedan svävar med oss. Det är ju inte fiktion. Det är inte det alltså.

Jag kommer ihåg när jag fick mitt första modem, det var på 9600 bara. Det var på början av 90-talet då och man tänkte WOW! Och det vara någon sa det att man kunde ringa upp en sådan ’modempol’ för i tiden då. DBS hette det ju, så kunde man ringa dit och ladda ner saker och sånt. Det var ju verkligen ’state-of-the-art’. Det fanns ju knappt internet då. Och se nu vart man har kommit på 20 år. Tänk 20 år till. Då är ju utvecklingen ännu ett steg vidare från objektorienterade då. Det måste ju komma något efter det.

Efter intervjun tackade vi så mycket för samtalet.

Bilaga E – Intervju 2: Inledande intervju med Intervjuperson B

[Fråga 1] Uppgiftssökare: *Vi har tänkt köra en lite mer öppnare intervju då. Syftet med vår uppsats är att jämföra om objektorienterad design är bra eller dåligt i dagens utveckling i mindre projekt med tanke på deadlines, smidighet, etc.*

Jag börjar med en lite generellt med att fråga vad det är för projekt du brukar arbeta med nu?

Informant: Ja, hur ska jag beskriva det nu då. Jo det vi har nu är ju en webbaserad applikation som började utvecklas för 5 år sedan. Sen så för ett och ett halvt år sedan så kom de fram till att vi förbättra det, utveckla det, gå vidare med den. Innan dess hade de väl någon diskussion, vad jag har förstått, huruvida de skulle vidareutveckla det eller inte. Det var några då som ”Nä, vi skiter i att utveckla. Vi har det vi har, vi ser hur länge det går”. Men till slut så blev det ju de, de som tyckte att det måste utvecklas för att hänga med, för att få nya kunder och sånt.. Som vann. Så det projektet..

På ett sätt är det ju bara ett enda projekt alltihopa som bara rullar på då. Vi har ju ingen direkt uttalad projektorganisation heller. Det blev liksom, en produkt som ska utvecklas.. Eller så, komma vidare med. Och man kan ju säga det fanns.. Innan jag kom in i företaget då, för ungefär ett och ett halvt år sedan, så kom där in en konsult. Så konsulten och företaget tillsammans, alltså styrelsen och konsulten.. Gjorde då sju steg som skulle gås igenom för vår produkt. Jag kan inte de sju stegen i huvudet så men, där fanns i alla fall en plan på vad som skulle göras. Sen nu så har det ju visat sig att säljarna inom företaget går mycket snabbare fram än vad vi hinner utveckla, eller som konsulterna hinner utveckla. Så just nu.. Av de här sju stegen så har vi inte gjort ’steg 1’ sen ’steg 2’, utan vi har gjort fem av de sju stegen när vi jobbar.

[F2] U: *Hur många utvecklare är ni på detta?*

I: Där är.. Vi köper in ett konsultbolag som består av två personer. En av dem lägger kanske 80-90% av tiden på det, medans den andre bara hjälper till då va. Sedan försöker jag.. Min roll är lite svår och oklar i detta men, jag försöker ju vara länken där mellan användare och programmerare då va, konsulten. Jag försöker då.. När någon då bestämt sig för den här delen i systemet, så försöker jag då snacka med de som har kontakt med våra kunder. Hur vill vi ni ha det? Hur ska det funka? Och sen försöka sammanställa det och sen snacka med konsulten som kodar. Och det gäller ju lite på andra hållet med när det är en konsult som ska koda. Han behöver ju information om hur han ska utveckla det. Hur ska det funka? Så då snackar han med mig, och berättar om det rent kodmässigt. Sen vänder jag mig till kunder då och pratar verksamhetsmässigt. Jag är lite såhär.. I mitten så att säga.

[F3] U: Hur brukar början av projekten se ut? Lägger ni upp någon planering för hur det ska se ut, jag tänker sammanhållning, talande klasser, etcetera?

I: Ehm, han kodningskonsulten, eller ska vi säga Janne? Han försöker ju komma dithän. Om vi säger att det är två personer som bara pratar verksamhet (som inte är datamänniskor), och så Janne, och så jag. När vi sitter i ett möte så brukar vi ju börja prata rent användarsnack. ”Man ska klicka här, och så ska det hända”. Ja, rent verksamhetsmässigt utan att snacka data. Sen så efter någon timme, efter flera timmar så brukar jag och Janne då försöka.. Så reser han sig upp och så börjar han rita då.. Han ritar lite klasser då, som vi ser det, för att han har ju inte det språket. Han ritar ju mer upp ”Så har vi en sådan här, och så har vi en sådan här, och en sådan ska kunna ha flera sådana”. Ja, han börjar snacka sådär att vem som helst begriper det. Men det är ju klasser han ritar upp.

[F4] U: *Nu vet jag inte om du vet detta men. Strävar han efter förståelse inom koden? Att koden är lite mer talande, beskrivande.*

I: Ja, jo det är väl en sådan typisk sak som man vill ha. Men som inte blir kanske. Nu jobbar ju Janne tydligen så att.. Vad jag har förstått i alla fall.. Att han gör.. Ibland är det ju så jättebråttom, så bara kodar man så att det funkar. Men sen så har han ju som mål att han går tillbaks sen och.. Städar i koden så att säga. Och då lägger han ju kommentarer, tar bort hårdkodade saker som kanske istället ska täckas av databasen. Så han gör ju det i flera steg..

Vi har kommit överens om att vi använder engelska som standardspråk, för att från början är ju systemet gjort på svenska. Journal.. Objekt som pekar på journal-objekt i databasen heter journal i koden. Vi försöker nu använda ’case-file’ och sådant då va. Men jag tycker vi försöker använda kodspråk så att det blir hyfsat läsbart.

[F5] U: *Hur svårt var det när ni kom in i projektet? Var det samma konsulter som innan, eller hade de svårt att sätta sig in i arbetet?*

I: Nej precis.. Det var nya konsulter. Men ja, jag tror att det var medvetna om hur det var. De satte sig några månader innan för att bara sätta sig in i koden. För att då få en uppfattning om hur lång tid saker och ting skulle ta att ändra och så vidare.

[F6] U: *Hur brukar storleken se ut på låt säga mindre moduler? Spelar storleken roll rent kodmässigt om man tänker kanske optimering, arvsbredd, antal klasser, eller det ska bara snabbt ut?*

I: Nja jag tror.. Tror säger jag för att jag kodar ju själv inte så mycket men, det är nog viktigare att hålla det, på bredden som du nu säger. För att Janne har ju varit med sen 80-talet och kodat så han har ju varit inne i det här träsket att man bara kodar på och till slut tappar man kontrollen över koden. Så han, han tycker det är oerhört viktigt att göra saker rent. Det är likadant med databasen, han.. I det som tidigare fanns så fanns.. Säg att det var 5 tabeller i databasen. Men så som han gör så kanske det blir 20 tabeller av det. För det ska liksom vara rent. Det ska vara en tabell för vilka typer av journaler som finns, istället för att lägga allt det i tillsamman med en annan tabell. Innan var det liksom en tabell.. Ett formulär som man då fyllde i blev en tabell. Men med Jannes sätt att hantera tabeller på så blir det då ja.. 10 tabeller som då hänger ihop på olika sätt. Ska man ändra någonting så gör man det i den tabellen där, en annan ändring gör man i den tabellen där, och så vidare. Och jag tror att han jobbar.. Koden blir ju i och för sig..

Det är en av skillnaderna jag har sätt. I det gamla systemet så var det jättemycket kod, men få tabeller i databasen. Men med Jannes sätt så jobbar vi tvärtom. Databasen blir jättekomplex, men koden blir då ganska kort istället. Han låter ju databasen jobba mer.. Och det var ju det.. Anledning till att han gör det är ju att det ska bli lättare att underhålla systemet senare.

Tanken är att, gör han sen ett gränssnitt så kan man ju mata in i koden hur systemet ska funka sen lägger sig de ändringar i databasen och påverkar på hur systemet jobbar. Istället för att man ska ha en programmerare som ska leta i koden.. Som ändrar i koden.

[F7] U: *Då är alltså förståelsen bra under driften. Ni gör det så enkelt som möjligt för er själva?*

I: Jo precis. Hela tanken är ju att det blir mer lättunderhållet senare. När han är färdig då..

[F8] U: *Precis. Tanken är väl att andra konsulter ska kunna komma in senare och förstå systemet?*

I: Ja exakt. Allra mest lyckat blir det ju om jag.. Jag kan hålla i det senare. Underhålla det så mycket som möjligt. Alltså jag som då är anställd inom företaget så att de slipper köpa in programmerare hela tiden så fort det är någon ändring.

[F9] U: *Allt eftersom projektet börjar ta form, har ni någon viss testning som sker kontinuerligt? Alltså, att ni gör något och så kollar du om det är bra. Eller gör ni en stor testning på slutet innan det kommer ut till kunden?*

I: Nä, det är mest småtestande hela tiden. Vi kör ju flera steg. Vi har.. Janne har ju en lokal miljö han utvecklar i. Sen har vi en testmiljö. Sen har vi en produktionsmiljö. Alltså, att kunderna använder produktionsnivån då. Så att vi går ju flera steg. Han utvecklar och testar, sen lägger han upp det på testmiljön och säger till mig att: ”Nu får du testa det här och det här och det här”. Jag vet ju vad som ska hända också. Och så sätter jag mig och provar det, och kanske någon mer anställd provar det.

Är det viktiga grejer så har vi även tillsatt en användare, alltså en grupp användare av kunder så att vi då kan lägga upp det i produktionsmiljön. Alltså, en begränsad del och säga till den här testgruppen att nu har vi släppt det här. Vill ni vara snälla och gå in och kolla så att ni tycker att det här är okej? Tycker då de att det är okej så släpper vi då det till skarpa kunder så att säga. Men det vi har märkt då också är att det är rätt många grejer som.. Alltså, det är rätt svårt att testa alla ’case’. Man tycker att man testat saker, men sen när man släpper det till slutkunder så märker man ändå att: ”Oj, det funkar inte som vi hade tänkt oss”.

[F10] U: *Du sa ju det innan att han Janne arbetar lite olika. Ibland så planerar han väldigt mycket med att rita upp vad han gjorde, eller så gick han in och gjorde det och gick tillbaka sen. Har ni märkt någon skillnad på effekten av att göra det? Alltså, vad som tar längst tid?*

I: Jo, det blir ju så att det tar längre tid när man gör snabbfixar. Alltså, om det blir något fel. Att det då utvecklarna utvecklar blir fel så måste man gå tillbaka och gör om det igen. Medan när man gör det från grunden och vet att det här ska tid, så tar det ju tid i början innan man kommit fram till hur man ska koda. Så kanske man kodar lite och så testar man och så kommer man fram till att: ”Nä, så här skulle det inte vara”. Det känns ju lite som att det tar.. Det blir ju.. Alltså, slutresultatet blir bättre när man närmar sig release. För då har man det genomtänkt. Medan, när det är snabbfixat så chansar man lite mer. Så får kunden då höra av sig: ”Du, det blev inte som jag tänkt”. Så får man dra det ett varv till.

[F11] U: *Jag funderade på det här.. Vi intervjuade en kille igår. Han skyllde mycket av det här strikta objektorienterade tänket på just de som jobbade under 80-talet när det var mycket mer viktigt med minne.. minnesutrymme och så vidare. Du tror inte att det kan vara lite av ett hinder som försenar projekten? Att man lägger för mycket tid på det kanske?*

I: Ja.. Jag vet inte om jag kan svara på den frågan. Men det jag tänker på det du säger är ju.. Som jag berättade så har ju Janne byggt upp det med en massa tabeller, och att koden ofta är inne och läser i tabeller. Men det jag märkt är ju att det till exempel blir långsammare. Jag vet inte om det påverkar att det är ett webbaserat system också. Så varje läsning ner i databas tar tid. Och det har vi märkt.. Man kan göra det som 1000 rader kod med väldigt avancerade beräkningar. Ser liksom mycket ut när man läser koden, men det tar kanske en tusendels sekund att exekvera. Men har man en 10 rader kod och man ska ner och läsa i databasen fyra gånger, så tar det ju mycket längre tid. Det har vi ju kommit på. Så där har vi försökt hitta lite lösningar så att man kan minska antalet läsningar, så att man ’cachar’ data och så vidare. Jag vet inte om det hade att göra med din fråga och så..

[F12] U: *Jo, just det att ni också bryter ut det centralt. Underlättar det driften mycket?*

I: Bryter ut centralt. Hur menar du?

[F13] U: *Alltså till databasen. Att ni kan underhålla centralt.. Att ni inte alltid behöver gå ut till varje klient..*

I: Jo, det gör ju underhållsarbetet mycket lättare ju. Vi gör det ju liksom på ett ställe. Behöver inte åka runt och skicka ut massa ’patchar’ och så.

[F14] U: *Hur sker just den driften? Med ’patchar’ och så. Måste ni åka runt till kunderna och applicera dem?*

I: Nej, alltså det är ju på webben vårt system ligger ju. Så vi bara stänger ner systemet en timme och så in med ändringarna och öppna det igen, så har alla det nya ju. Det är ju en väldig fördel. Vi samarbetar med företag X, som har ett system som lokal tjänst. De har ju ett helt annat sätt att arbeta på. Där.. Deras programmerare har en deadline när de måste vara färdiga. Och efter det ska det testas. Och sen när det har testats färdigt så gör de en version och så skickar de ut den till alla kunder. Sen är det ju inte alla kunder då som har kunskap om att installera det så då får de hålla på och åka runt och hjälpa kunderna installera det. Så det gör de kanske en två, tre gånger om året. De kan liksom inte släppa nyheter oftare än så. Vi kan ju göra det varje vecka om vi vill.

[F15] U: *Det blir rätt stort arbete för dem då?*

I: Jo, just när man är ute på någon sådan träff med någon kund som efterfrågar någon förändring. Så säger hon då på företag X: ”Ja, jo. Det kan ni ju få om en två år”. Jag menar, vi säger ju: ”Ja det kan ni ju få om ett par veckor”.

[F16] U: *Men ni kör inte efter deadlines då?*

I: Nej, inte så strikt. Alltså , vi sätter ju mer upp det själva ibland för att kunna säga så till kunderna liksom. Ja alltså, när vi samarbetar med företag X så måste vi ju ’synca’ det med deras deadlines och så. Men vi hade en deadline nu för några veckor sedan. Men då gjorde vi ju färdigt den biten som företag X märker av. Sen kunde vi liksom fortsätta jobba inne i vår kod ändå ju. Det räcker att vi har det färdigt till den dagen deras kunder har installerat sina förändringar ju.

[F17] U: *Tror du att det här att ni har mindre tidspress.. Att ni kanske kan planera mer designmässigt innan ni börjar koda? Om det finns något sammanhang där? Till exempel om ni haft en stark deadline så kanske ni hade fått programmera snabbare för att få ut det till kunderna?*

I: Jo, man märker ju den skillnaden. Vi har ju en VD som trycker på oss. Han är ju mycket ute så där och träffar kunder. Och så lovar han att: ”Jo, det ska vi ha fixat om två månader”. Även om det inte är deadlines så.. så skapar han ju en stress ibland, som gör att det blir det här snabba sättet att utveckla på ändå. Alltså, att det att man lovar saker till kund. Janne och jag som sitter med det dagliga, vi kanske mer kanske säger att det vore mer rimligt att kunden skulle få det om sex månader, men då har ju VDn sagt två månader. Så då får man sätta sig ner och prioritera; vad ska vi göra, så gör man vissa delar rätt snabbt. Och så vet man att resten som också borde varit gjort får vänta lite.

[F18] U: *Blir det ofta då att ni får gå tillbaka till det ni gjort.. Det ni halvstressat ut. Att det kanske inte alltid blir så bra?*

I: Jo, man får ju backa tillbaka ett steg där ja. Det höll ju på så.. I höstas höll det ju på så hela tiden att det kom in nya grejer hela tiden som måste göras, så att ingenting blev färdigt. Alltså, det bara växte det man skulle göra hela tiden. Så någon gång i vintras gick vi in och bestämde att nu måste vi bestämma oss för vad vi vill göra färdigt, så att kunden märker någonting. För vi snackar om att vi utvecklar mycket, men kunden märker ingenting. För ingenting blir färdigt. Så då bestämde vi oss för att dessa saker skulle vi göra. Så blev det då en lista på 100 punkter. Och då tänket vi att: ”Ok, när de 100 sakerna är färdiga så släpper vi till kund”, men även det fick vi ju revidera, så det blev ju 70 av de 100 punkterna som blev färdiga så gick det till kund. Och nu sitter vi och försöker göra färdigt de sista 30 punkterna.

Det svåra är just det att vi tycker att vi ska göra färdigt det, men då tycker ju andra att det ju är färdigt och att det är dags att släppa nästa grej. Det känns lite så.. Jag har känslan att man aldrig blir färdig. Och jag kan tycka att det är lite jobbigt att den känslan att nya grejer som ska utvecklas fyller på snabbare än man hinner göra färdigt. För mig är det en ny grej eftersom jag kommer från kommunvärlden. Men Janne som har utvecklat, han är cool lugn. Han berörs liksom inte. Han vet att så är det.

[F19] U: *Är det svårare att implementera de här 30 punkterna än om ni gjort det med en gång? Just att ni behöver gå tillbaka efter ’release’ och lägga till de här 30?*

I: Ja, så kan det ju vara. För när man håller på med någonting så har man ju det färskt i huvudet. Släpper man det och sen gör något annat ett tag.. Sen när man kommer tillbaka till det så tar det ju lite tid att få in det i huvudet igen. Vad man tänkte. Så det blir nog merarbete på det viset.

[F20] U: *Du som är nyutexaminerad, har du några stora skillnader mot hur vi har lärt oss?*

I: Jag kommer ihåg första månaderna när jag kom ut på jobbet, att jag var fascinerad över att vi hade lärt oss något som man faktiskt hade nytta av. Jag menar, just det här att jobba med UML-diagram är ju väldigt användbart. Det finns liksom överallt. Det är bara att läsa det. Jag har jobbat mycket med databas. Både att hjälpa Janne att fylla på databasen, men även när kunder ringer och något inte stämmer, så går man in i databasen och ändrar. Och det tycker jag också det vi lärde oss där att skriva SQL-satser.. Det är ju skitbra och har mycket nytta av. Även om.. Jag kommer ihåg att de sa det på utbildningen att typ 90% av de SQL-kommandon som skrivs är enkla SELECT FROM WHERE. Och det stämmer ju. Men det är ändå bra att veta att man kan göra mer avancerade sökningar om man vill.

[F21] U: *Hur väl stämmer de här.. Jag kommer ihåg första systemutvecklingskursen vi läste. När man gjorde rika bilder och användningsfall och UML och alla de här stegen. Stämmer det väl överens om hur det ser ut i ert projekt?*

I: Ja på ett sätt tror jag det. Sen kanske man inte använder just rika bilder. Vi jobbar inte så att man måste använda det och sen använda det. Man kan nog använda det lite fritt. Vilka metoder man vill, eller sätt man vill. Men jag tycker det stämmer. Man måste nog ändå göra det. För du har någonstans en kund eller en användare som inte har en aning något om vad en databas eller kod är, utan som bara sprutar ur sig: ”Så här ska det fungera”. Någonstans måste man ändå få ner det med det här strukturerade. Och kunna rita för sig själv så man fattar hur man ska koda det, men ändå kunna berätta tillbaka: ”Är det så här du har tänkt dig?”.

[F22] U: *Sen efter detta steget. Brukar ni använda er mycket av mönsterlösningar och färdiga patterns?*

I: Ja, det är ju sådant som jag inte vet. Det är mer Janne som sitter med det. Så jag vet inte hur mycket han använder sådant som är officiella mönsterlösningar, eller om det är mycket som är hans egna erfarenheter, som

han brukar göra. Det vet jag inte. Men jag vet om att han brukar snacka om att det gamla projektet.. Han vill bryta ut så mycket som möjligt så att det inte upprepar sig 16 gånger liksom.

Efter intervjun tackade vi så mycket för samtalet.

Bilaga F – Intervju 3: Efterföljande intervju med Intervjuperson A

[Fråga 1] Uppgiftssökare: *Du skapar ju kfm flera gånger. Du har ju massa metoder här. Är det mycket 'copy/paste' eller?*

Informant: Det är mycket 'copy/paste'. Det är det. När vi började med det.. Nu snackar vi timmars utveckling. Så gav jag dem en tid på åtta timmar tror jag, åtta nio timmar. På en arbetsdag, och så skulle jag också lära mig hur det fungerar. Då blir det lite.. Det är ett gammalt system. Det är svårt att ta reda på exakt hur det fungerar. Så testar man lite, så skapar man en klass, så kallar man på en metod och hämtar lite. Sen sitter man och funderar: "Hur ska jag få det här att fungera om jag ska göra det här om och om igen?".

De har ju också en sådan regel att man bara får skicka 3000 personnummer åt gången. Så väntar man lite, så får bara man bara skicka 3000 igen. Jag vet inte varför de har det. Så får man max skicka 15000 per dag. Det är lite sådana regler. Så därför blev det lite så. Det blev lite 'copy/paste'.

[F2] U: *Det är kanske lite samma sak. Du har ju klassen kfm som gör allting. Alla SQL-grejer och filgrejer..*

I: Men du.. Och jag vet att om man ska göra det på rätt sätt så, men vad gör det för skillnad egentligen? Alltså rent prestandamässigt. Nu är det ju ett väldigt litet projekt, men det gör ju ingen skillnad egentligen. Det som är dåligt där är ju sådana saker som 'connect-metoden', som man anropar om och om igen. Det är ju jättedumt.

Sen har man ju inte följt någon objektorienterad strategi. Jag har ju liksom inte skapat objekt för allt som kan vara objekt. XFR kan ju vara ett objekt, men det hämtar jag. Men än en gång, det pratade vi om innan, man ska göra någonting så börjar man någonstans. Och just de här 'command-programmen' som inte har någon 'GUI', alltså inget grafiskt interface, då blir det ännu lättare att man bara kör på.

Och lite av den approachen som ni ser där är ju mer ett scriptat språk. Ni vet ju hur man jobbar i 'javascript', man skapar en 'main metod' och så kallar man på 'subrutiner', till exempel om ni vill göra någonting, skicka tillbaka någonting. Det blir ju.. Det blir liksom ett mellanting mellan ett objektorienterat språk och ett scriptspråk, som man då trycker ihop för att få någonting.

[F3] U: *Det är inget projekt ni kommer gå tillbaka till och bygga ut massa? Så att det inte finns något behov av utbrytbarhet?*

I: Absolut inte. Det jag skickade till er ju intressant på det viset att det är ett 'user interface'. Ett 'user interface' som i grund och botten gör en enda sak. Den skickar upp ett personnummer, och det personnumret får du tillbaka av kronofogden om den här personen är skuldsatt. Hur många 'A-mål' och 'E-mål' och anmärkningar de har hos Kronofogden om det pågår utmätning. Det är egentligen det enda det gör.

Sen att det blev så komplicerat är bara för att de har ett så gammalmodigt system. Och den dagen de gör om sitt system så kastar vi bort detta interfacet. Då byter vi till ett traditionellt 'web service', där du bara skickar in ett personnummer och får tillbaka alla 'properties' du vill ha. Och då behöver man inte mer.

[F4] U: *Det är ju mycket så här 'button1' och sådana grejer. Ingen anledning att mecka med det kanske? Jag just på namngivningen av vissa av de här sakerna. Det kanske inte är så talande för någon som skulle kommit utifrån och kolla på koden liksom.*

I: Än en gång tror jag det blir lite så med storleken alltså. Du skapar en knapp då och sen kopplar du ett 'event' till den då, som du 'triggar' då när du klickar på den. Det är ju så enkelt att se liksom, vilket 'event' som tillhör liksom. Man behöver inte köra någon 'naming-standard' rakt igenom liksom.

[F5] U: *Sen de här lokala.. eller inte loka variabler. Du kör ju 'underscore' . Ibland gjorde du det på lokala variabler och ibland inte. Är det bara som du missat eller?*

I: Det är nog bara som jag missat. Jag tror det har lite med det här vi pratat om innan. Jag tror utveckling blir lite hur man är som person liksom. Jag är nog väldigt energisk. Jag vill ha saker gjorda väldigt fort. Och är kanske.. Vi tar den här liknelsen med en hantverkare. En gång så renoverade jag mitt hus, så hjälpte en kompis mig. Så skulle vi bygga innervägg. Och ni vet hur en innerväg ser ut, va? Man har ett regelverk då av trä, oftast '45 70' typ träreglar. Så sätter man dem 30 till 60 centimeters mellan rum då, beroende på om det ska vara en bärande

vägg eller en vanlig vägg. Och sen kanske man sätter någon form av isolering emellan om det ska vara lite ljuddämpande, och så sätter man gips på utsidan.

Så stod jag då och kollade på den här killen. Så byggde han den väggen exceptionellt bra. Du vet, han 'lodade' varje regel, satte trä mellan varje regel så att de inte skulle kunna böja sig på olika håll och gjorde det jättefint. Han förborrade. Och sen kommer jag dit och bara smäller på gipsen, spacklar och målar. Men man såg endast det yttre av väggen. Och funktionen av väggen hade ju ingen som helst betydelse.

Och sen nästa dag så träffade vi en annan kille som byggde den här väggen. Han bara smällde upp väggen. Det tog nolltid liksom. Han bara gick på och spikade och spikade och skruvade och bara spikade allting, smällde på gipsen och var klar. Han gjorde det kanske på en fjärdedel av tiden. Men väggarna såg identiska ut när man tittade på dem utifrån. Lite så är det med utveckling också.

[F6] U: *Det skapar liksom inget kundvärde?*

I: Det skapar inget mervärde i mindre till medelstora projekt. För har man missat någonstans eller gjort någonting lite halvdant.. Om programmet fungerar och gör vad det ska och man inte gjort fatala fel liksom. Att inte göra det här 'connect' om och om igen, och inte använda de här 'SQL-command' om och om och om igen.

Det är ju inte så supersmart. Men å andra sidan. Du ska göra någonting. Du ska kalla och hämta något från en databas. Så skapar du det 'statementet'. Sen 'copy/pastar' du den funktionen om och igen. Det är ett litet hastverk kan man säga. Men om du gör ett projekt i den storleksklassen så.. Jag vet inte vad det riktigt gör för skillnad egentligen.

[F7] U: *Det finns ingen risk att om man sitter och gör många sådana här mindre projekt och utvecklar på det sättet, att det kanske följer med till när man börjar på ett lite större projekt?*

I: Absolut. Ibland börjar man med något litet som ser mycket värre ut än det här projektet. Som från en metod hämtar 'SQL' och sen gör det och det. Och upprepar kod om och om och om igen liksom. Saker som är helt idiotiska.

[F8] U: *Nja, jag tänkte mer som så att om du har jobbat med tio extremt små projekt och sen ska över till ett större projekt, att ditt beteende kanske ligger kvar? Eller du klarar av att ställa om till ett ltie större projekt då?*

I: Jag tror att jag ställer om lite. Det har liksom med tid och vad för förväntningar en kund har liksom. Jag tror att man ställer om lite, men man ställer ju inte om helt liksom. Om jag jämför med ett projekt som följer skolboken så skulle jag aldrig göra så. Och även om jag gjorde ett jätteprojekt så skulle jag inte göra så. Och det är ju kanske för att man arbetat mer med.. Jag kommer ju då från 'C' och 'C++', som kanske är en lite annorlunda värld från dagens objektorienterade språk. Och jag tror ju att jag gjort det mycket bättre. Jag hade ju inte kastat ihop det så här. Men jag hade ju inte gjort det exakt som skolboken säger.

[F9] U: *Vi hittade ju en annan grej. Du hade skrivit kommentarer till SQL-metoder. Och kommentarerna tyckte vi inte var mer talande än själva funktionsnamnet var.*

I: Har du något exempel?

[F10] U: *Du hade väl kanske någon som sa 'GetPersondata' eller någonting. Och så i kommentaren: "Hämtar persondata från databasen".*

I: Det var nog mycket slarv. Och sen mycket det här med kommentarer .. Alltså jag är ju sådan som person.. Alltså det handlar mycket om hur.. Alltså, jag träffar många utvecklare. Alla har ju sin egen stil. Vissa är slarviga, vissa är jätteduktiga, vissa är duktiga på att kommentera och så vidare. Jag är ju en sådan person att så fort jag löst problemet så tycker jag att jag är klar. Då går jag vidare till nästa steg. Ibland: "Nä jag får väl skriva ihop någonting där". Så kladdar jag ihop någonting. Då blir det liksom att jag måste skriva någon kommentar för varje metod för att jag ska ha skrivit någonting. Och det är ju helt rätt. Varför ska man skriva det? Jag skulle ju bara inte skrivit någonting alls. Men det är bara slarv.

[F11] U: *Bättre att skriva än att inte skriva alls kanske?*

I: Ja jag vet faktiskt inte. Ibland så.. Jag tycker det är jättesvårt det där med att kommentera kod. Det är ju precis som du säger.. Det finns ju jättemånga metoder som nästan talar för sig själv. Man vet ju exakt när man tittar på den, vad ungefär den här gör. Egentligen ska man ju bara kommentera saker som är avvikande.. Som är annorlunda.. Som.. Men det är ju samma med projekt X, så står det ju ”Det här är ett person-objekt. Hämtar information om en person.” Jaha, men det fattar ju jag också..

Och sen har de ju bara gjort den här person-objekt, och sen är det ju bara ’first name’ och ’last name’ bla bla bla.. Och så bara hämtar och lämnar den någonting här då. Det här är ju väldigt bortkastat. Vad ska jag med detta till? Det är ju ingen direkt manipulation eller sådär. Det är ju bara hämta, eller släng, en sånhär privat variabel.. Så jag tror.. Alltså man kan ju gå åt båda hållen. Man ser ju sådana som kommenterar precis ALLT.. Eller man kan liksom bara rent kastar ihop det.

[F12] U: *På tal om det med variabler, du manipulerar ju rätt så mycket i de här ’get-’ och ’set-funktionerna’. Är det för att tjäna tid?*

I: Nja, det är ju inte bara för att tjäna tid. Det är ju det, också att.. Om du gör ett objekt, en klass till exempel.. Kalla den personal.. Så i databasen har du då förnamn, efternamn då ju.. Så i funktionen vill du ha ’full name’, det fulla namnet då ju. Man kan göra på en massa olika sätt då.. I databasen tar du då SELECT förnamn, mellanslag, plus två tecken, mellanslag, efternamn då va. Då får du ju.. Och så kallar du det fältet någonting.. Då får du ju ett fält tillbaka i SQL-satsen som är ’full name’ då.

Men om du gör det till exempel i din klass, så får ju ju en massa fördelar.. För det första. Om du använder klassen på 20 olika ställen, och så kommer någon och säger ”Du, jag vill faktiskt att det ska vara förnamn, mellanslag, efternamn.” så har du ju alltid det på ett enda ställe som du kan hämta ifrån. Det blir inget jättekonstigt egentligen men.. Då finns det ju på ett ställe, och du hämtar det därifrån och då blir det alltid samma ju.

[F13] U: *Tanken med det här extremt strikta objektorienterade är ju att man hellre kanske ska bryta ut det till en egen funktion då.*

I: Ja, man kanske krockar lite i det där tankesättet. Jag vet inte riktigt vad jag skulle tjäna.. Vad skulle jag tjäna på det. Vad säger egentligen boken om vad jag skulle tjäna på det? Det förstår jag inte riktigt.

[F14] U: *Mm, jag tänkte på det här med ’file locations’. Du har ju brutit ut det till en settings-fil, men har ändå på vissa ställen skrivit ut den fulla strängen i koden. Har du kanske tänkt att använda det flera gånger och därför ligger den även i koden?*

I: Hmm, jag vet inte exakt alltså.. Det finns ju den här ’System.ConfigurationManager’ för att hämta saker från den här ’App.Config’ då ju. Den använder vi ofta då ju.. Med namnet då va.. Den setting jag vill hämta så att säga.. Jag kommer inte ihåg riktigt men.. Där kan man ju göra på ett mycket bättre sätt egentligen. Du kan ju skapa en ’AppGobal’ till exempel. Och i ’AppGlobal’ så när du startar programmet så hämtar den alla settings och lagrar dem tryggt där. Så slipper du gå ut där.. Hela tiden och hämta. Så att, man kan ju göra på en massa snyggare sätt och det..

Men det är ju hela tiden tanken man.. Man.. Nja, jag kunde ju gjort det snyggare men.. Om du börjar med ett projekt och någon säger så här: ”Du. Det enda du ska göra är att hämta de här 15 användarna här”. Man ska då göra ett uppdrag som ska ändra i deras system.. Allting är då i stora bokstäver, ’uppercase’ på allting så att säga. Då vill de ju att vi ska hämta ut alla namn, sen så ska vi då sätta ’uppercase’ på första bokstaven då va.. Versaler då, så det ser mer trevligt ut så att säga. Har man bara stora bokstäver så ser det ju ut som de skriker. Speciellt när man skickar brev.. Och sen.. Vi gör ett program som bara tar in det. Tar första bokstaven, gör de andra bokstäverna små, punkt, och ny stor bokstav igen då va.. Jätteenkelt då va.

Hur skulle ni då göra det? Skulle ni sätta er ner och starta ett helt objekt, sätta upp en klass, ett diagram, ’configurationStrings’ som hämtar från ’App.Config’ och hålla på med det i två, tre timmar innan ni ens börjar.. Jag har ju till exempel bara skapat en mer scriptvariant som hade *TOOSCH* hämtat in all information, så att jag har all information då.. Sen hade jag skapat ett som skickar tillbaka det bara. Med en ’foreach’ då till exempel, och sen är det färdigt. Jag tror det är väldigt, väldigt stor skillnad på små och stora projekt då. Det blir lite mer praktiskt liksom.

Det blir mycket.. Jag tror ju på det här med ’throwaway code’ som, som, som kanske är ett skällsord för många.. Men jag tror på det alltså. Du skriver lite kod, sen kastar du bort det. Det är som en skiss liksom. Som en skiss på någonting.. Det funkar bra.. Som en skiss på ett program som en kund gjorde på mötet före er.. Tar han den o sparar i en pärm och lägger undan? Nej det göra han ju inte då va. Den kastar han ju bara bort. För han har gjort jobbet, han har förklarat för oss vad han då ville att vi skulle veta.. Så han var nöjd med detta.. Han ville att vi skulle hämta ut allt ur systemet, ändra till liten bokstav, köra tillbaka det.

Ja, nu har jag gjort detta. Jag sparade ju kanske det någonstans, men jag satt ju inte flera timmar och funderade hur det skulle funka. Det känns ju inte realistiskt.. Jag tror att ni nog ska snegla lite på scriptspråk, att ni ska ha med det i er avhandling lite. Vad var tanken bakom dem? Och så vidare.. Finns ju en massa negativa saker bland dem.. Folk överanvänder ju scriptspråk.. PHP är ju ett klassiskt exempel på scriptspråk så att säga. Facebook är ju PHP till exempel.

[F15] U: *Vi har ju tittat lite på hur det såg ut innan objektorienterad programmering, hur det kom till. Det där med till exempel strukturell programmering, och så vidare.*

I: Ja precis. Det är ju intressant och så..

[F16] U: *Om vi går vidare på det här med felhantering. Ni skickar ofta tillbaka bara 1:or och 0:or utan någon direkt förklaring på vad det är, och så vidare. Vad beror detta på?*

I: Ja det är ju väldigt intressant faktiskt. I detta projektet, hur många människor tror ni det är som använder programmet? Vad skulle ni gissa på? Det är alltså en enda person som faktiskt använder programmet. Den här personen har inga som helst IT-kunskaper, eller jo kanske vissa men.. Det är ingen utvecklare, ingen IT-person precis.. Utan en administrativ person som arbetar med att [otydligt]. Vad jag än hade returnerat, eller vad jag än hade skrivit så hade inte den här personen förstått så mycket.. Fått ut så mycket av det.

Så därför blir det ju att man.. Om man arbetar på det här sättet så blir det ju ändå att man går på den här script-varianten.. Man bara skickar tillbaka någonting.. Det funkar inte liksom. Exakt varför det inte funkar tillhör ju också storleken. Vad skulle jag göra med information jag fick tillbaka? Det funkade inte för att XYZ så att säga.. Skulle jag visa det, eller skulle jag då använde den för att då göra någonting i programmet? Då blir det ju då helt plötsligt ett större projekt..

Felhantering generellt är ju.. Jättesvårt.. Men.. Hade jag haft ett lite större program som skulle användas av 50.000 människor så hade jag ju självklart lagt ner enormt tid på att folk ska förstå vad felet är och man skulle haft en kundtjänst som tog emot felet då va. Man kanske även skulle haft en kod då va, så att man kunde slå upp vad felet är.. Som Microsoft då va.. Slå upp, var är man i processen, och varför, och så vidare. Men än en gång.. En enda person som använder programmet.. Kanske, kanske två om vi har supertur.. Det finns ju inget skäl alltså.. Det är den enda gången man återvänder till den här koden.. Om det då inte finns något skäl. Varför då göra det bara för att det är god sed?

[F17] U: *Sista frågan.. Om du hade få gjort systemet igen, hade du designat det likadant då?*

I: ..... Nästan, det tror jag. Och varför.. Om du nu frågade varför.. För det första, det är ju bara en person som ska använda det. För det andra så är det ett ’user interface’ till ett uråldrigt system då va.. Jag kan ju lägga hur mycket tid och hur mycket energi på det här lilla, lilla programmet.. Lägga dagar på att göra det ännu bättre men.. Det är ju ändå bara en del i en länk.

Är den ena delen då jättedålig, gammalmodig, och fungerar halvdant så kvittar det ju hur mycket tid man lägger från sin sida. När de skickar tillbaka sina felkoder så står det ju bara något jättekonstigt.. Någon konstig kod som jag inte ens vet vad det är. Jag ringer tillbaka till dem då, för att ta reda på vad det är.. Allting är gjort på [otydligt]. De kör det här [otydligt] som hämtar hem information om 27.000 personer, varje vecka. De använder det varje vecka, och det funkar, mer eller mindre varje vecka.

Den enda gången det inte funkar är om de ändrar någonting.. Nummer.. De måste ha ett specifikt nummer varje gång de skickar upp.. Är det någon annan i företaget som då tar det här numret så blir det fel i sekvensen. Annars funkar det.. Så att.. Småsaker kanske jag hade ändrat men.. Inte enormt mycket, det tror jag inte.

Efter intervjun så tackade vi för samtalet.

Bilaga G – Intervju 4: Efterföljande intervju med Intervjuperson B

[Fråga 1] Uppgiftssökare: *Vi har delat upp intervjun i tre olika delar som tar upp områden som kan påverka utvecklingen, utifrån vad vi då har hittat genom dokumentundersökningen. Om vi börjar med att; flera klasser var utan instansvariabler och höll bara i olika metoder. Vad kan detta bero på tror du?*

Informant: Hmm.. Har du något exempel kanske?

[F2] U: *Vi stötte ofta på det i samband med olika 'partial' klasser, att ena delen hade variablerna och andra metoderna.*

I: Hmm.. Ja jag vet inte exakt men.. Jag tror det beror på att.. När man drar datatabeller.. Nä, vad heter det.. In, via 'LINQ'.. Om du drar en tabell och så. Då skapas ju automatiskt kod och så. Med alla instansvariabler. Vill du sen hålla på att manipulera det så skapar du en 'partial' klass där du lägger egna definitioner, egna metoder.. Det kan möjligen vara en orsak till att det blir 'partial' klasser där det bara ligger en massa metoder. Man har redan automatiskt skapat de andra, och så vill man lägga till grejer.

[F3] U: *Det var väl främst klasser som ärvde från den här 'BusinessLayerBase'. En dll-fil som ni hade, i vilket det fanns alla de här behandling, och så vidare.*

I: Ja just det. Det är ju då den gamla, ursprungskodens motsvarighet till 'LINQ' där kan man säga. Alltså 'BusinessLayer' är då själva.. Där man har lyft in tabellerna då, så att säga.

Just nu finns det ju två olika.. De första kodarna jobbade då med de här 'BusinessLayer' och en massa 'tables' och sådana grejer. Han som kodar det mesta nu.. Janne då va.. Han kör ju en massa 'LINQ' frågor istället. Så det finns ju alltså två olika vägar in i databasen då kan man säga.

[F4] U: *Jag vet att ni nämnde det i förra intervjun. Det här med MVC, att tankesättet i stora drag saknas nu. Vad kan detta bero på tror du?*

I: Ja, jag hade ju en diskussion med Janne om det precis när jag hade börjat.. Hur jag tyckte vi då skulle bygga det.. Då nämnde jag det här just med MVC. Jag ritade upp hur vi då hade fått lära oss på hur man tänkte. Men han tyckte då att de här LINQ-klasserna som automatiskt skapas.. Det tycker han då är 'controllern' liksom. Han försöker ju bygga koden så att själva aspx-sidorna inte innehåller så mycket logik, utan att man slänger ner logiken till de här LINQ-klasserna. Så vi har haft en diskussion om det och.. Vi försöker att göra.. Försöker att flytta bort så mycket logik som möjligt från själva gränssnittet.

[F5] U: *Är det något som då har legat kvar sen tidigare? Jag vet att vi, på vissa sidor, kunde hitta sql-anrop i asp-sidorna. Är det något som är pågående nu, att få bort detta?*

I: Ja precis. Alltså när man tittar då i hela.. I alla kod då så att säga.. Då är det kanske 10-20% som då Janne har gjort. Resten är då det gamla som ligger kvar.. Så det finns ju väldigt mycket kvar då.

[F6] U: *Vi kom in på det lite innan om den här 'BusinessLayerBase', att många metoder och att många klasser ärver ifrån den. Använder alla dessa klasser metoderna, eller är det en genväg ni har tagit när ni har programmerat?*

I: Hmm.. Jag är inte helt säker där faktiskt. Men tror som sagt att det är mycket sådan auto-genererad kod.. Sen så står det ju ibland.. Visual Studio lägger ju in sådan här auto-genererad kod.. "Akta er för att gå in och ändra här". Den dyker upp lite överallt tycker jag.

[F7] U: *En annan sak också. Varför används just språket VisualBasic?*

I: Ja det var väl främst för att Janne kunde VisualBasic bäst. Vi hade också en diskussion om det här när jag började då.. "Ska vi inte passa på att byta till C#" sade jag då. Men Janne hade ju då kodat i VisualBasic i många år.. Han hade kunnat tänka sig göra det då va, men.. Han hade ju fått lära sig en massa nytt då, och med tanke på att allt redan var i VisualBasic då så "Okej, vi fortsätter med det".

[F8] U: *Något annat vi lade märkte till var att kod upprepades på många ställen, till exempel personnummervalidering. Är det också något som ligger kvar sen det gamla, eller det har något syfte?*

I: Hmm, jag tror faktiskt inte vi är medvetna om allt det där. Utan det är.. Det är ju lite samma sak där att vi har den känslan.. Hos vissa andra grejer har vi ju sett att det ligger logik i fyra olika aspx-klasser.. Samma logik på många ställen.. Så en av de anledning till att vi försöker lyfta ner logiken.. Samla det på ett ställe så att säga. Men vi sitter inte och aktivt letar efter detta när vi utvecklar. Utan när vi gör det nya, så gör vi det på detta viset. Målet är ju att sakta men säkert byta ut all gammal kod till det nya.

[F9] U: *Vi lade märkte till att ni bröt ut delar av asp-sidor som användes ofta, till kontroller. Vi kan tänka att detta gjordes för att slippa att just upprepa koden.*

I: Ja det är nog rätt så viktigt när det blir så stora projekt av det. Det blir omöjligt att hålla reda på att ställen annars.

[F10] U: *Vi såg även att vissa property-metoder hade mer funktionalitet än att bara lämna och hämta, vilket egentligen bryter lite mot det här med ”god sed”. Vad kan detta bero på tror du?*

I: Ja just det. Det har jag också diskuterat med Janne. Han tycker att sådana metoder.. Där man har ett objekt och ska fråga efter en egenskap.. Även om det inte direkt är en variabel så tycker han att man lägger det som en property ändå. Även om man sedan går in i andra tabeller och letar efter svaret så tycker han.. Är det liksom en.. En egenskap där man inte behöver skicka in någon inparameter så lägger man det som en property.

[F11] U: *Ja precis. Det sker ju även när, till exempel ni ska hämta ett datum, så göra beräkningar i propertyn om kanske vilket land det gäller, etcetera. Kan det vara föra att automatisera en del?*

I: Ja, det är ju säkert så att.. Från det stället där anropet sker.. Att det anropet ska bli enklare, mer lättläst liksom.

[F12] U: *För att gå vidare in på det här med drift. Vi hittade på många håll hårdkodad information, som till exempel länkar, databaskopplingar, etcetera. Är det också något som ligger kvar sen det gamla systemet, och hur påverkar det sedan om till exempel webbhotellet flyttas?*

I: Ja det kan ju säkert vara som ligger kvar sedan jättelänge som vi inte ens är medvetna om att de ligger där. Jag vet att Janne jobbar på det viset att han.. Vissa sådana här texter.. Alltså, han börjar med att tillverka bara för Sverige. Då kan han lägga in hårdkodat, en sträng för svenska. Sen markerar han upp det, och senare går han tillbaka till det och ändrar det till en databasfråga som kollar vilket språk det är som anropar. Han lägger medvetet kvar vissa hårdkodade grejer som han sedan försöker hinna med att städa undan i efterhand.

[F13] U: *Vi har sett att ni har en sådan där Web.Config i vilket ni kan lyfta ut connectStrings och så vidare, men ni använder den bara halvt ut. Är det något du känner till varför ni inte utnyttjar den helt?*

I: Nej det är inget jag känner till faktiskt.. Tyvärr.

[F14] U: *Vi upptäckte i er kod bland annat en ’enumerator’ som kollar någonting med hur många familjemedlemmar man enligt lag.. Alltså någonting där man beroende på land hade rätt till ett visst antal familjemedlemmar. Detta var hårdkodat. Vad händer liksom om de lagarna ändras?*

I: Då gäller det att leta upp alla kodställen den koden finns på. Nä, men det är typ sådana grejer vi vet om, som vi försöker städa efterhand som vi stöter på det.

[F15] U: *Är det något som varit ett stort problem innan? Att ni inte brutit ut saker tillräckligt. Rent generellt sådana här grejer?*

I: Ja, jo det kan man säga. Det var väl det som de tjafsade om då för ett och ett halvt år sen, när ägarna av styrelsen snackade. Där några ville utveckla programmet, och vissa tyckte: ”Nej, stopp och belägg. Nu får vi sluta utveckla och bara sälja det så länge det går”. Och det är väl de som kodat det som tyckte det sista då. Nä, men ärligt talat så var det så. Att en av ägarna hade varit med kodat. Och när de sedan bestämde sig för att fortsätta utveckla det så såldes de ut. Så bytte vi konsult, då till Janne då, som sitter och utvecklar och tänker på att göra kod så att det går att utveckla i framtiden.

[F16] U: *Är det något som tar merparten av er tid? Att sitta och bryta ut och städa jämfört med att lägga till nya funktioner?*

I: Alltså, just nu gör det inte det. Men när jag började var vi ju inne i ett ganska så hektiskt skede, när vi hade fullt upp och fixa fel. Alltså folk ringde att här är det fel då. Systemet brakade ihop och så där. Så förra året var det över sommaren och fram till augusti någonting, så var det hela tiden att jaga ifatt.. att få systemet stabilt. Sen lyckades vi väl med det, plus att jag då kunde ta över vissa av konsultens arbetsuppgifter, som att gå in direkt i databasen och korrigera fel, som kunder har gjort.

Så det känns som att i höstas någongång så kunde Janne börja med att jobba framåt och förbättra kod.. att göra ny kod och ta bort gammal. Så det känns som att just nu.. Alltså, vi sitter inte och letar dumheter i den gamla koden, utan vi gör så att vi bestämmer oss för en bit. Till exempel hur journalen är uppbyggd. Nu tar vi det. Nu gör vi något nytt, och så kan vi sen ta bort det som är gjort sen tidigare. Och det blir ju jobbigt för Janne, för han får ju hela tiden utveckla så att det gamla fortfarande fungerar i det nya han gör, men.. Vi har valt att göra det för att kunna ta det bitvis liksom.

[F17] U: *Fel som uppstår, förutom enkla fel som kan korrigeras direkt i databasen.. Alltså, är det saker som tar lång tid att fixa?*

I: Nä, att det helt har klappat ihop har jag inte varit med om. Det kan vara sådana enkla fel som att man stöter bara på just det felet när man är inne på den länken och man ska göra en viss grej. Då låter vi systemet gå och sen letar vi fel via testsidan då, som inte berör de som köper tjänsten då. Så letar vi upp felet där och korrigerar det. Sen så stänger man bara ner systemet en timme och så lyfter man in en ny version. Så sätter man på det igen och så fungerar det. Det blir väldigt lite sådana driftstopp, det blir det.

[F18] U: *Kundrelationerna påverkas inte negativt?*

I: Nä, det är ju om.. Vi lade ju rätt mycket krut i höstas på något som heter bakgrundsvariabler. Det är sådana långa frågepaket med hundra frågor man ska svara på; Var går barnet, och var de har för relation till sina föräldrar och så vidare. Där var det massa konstigheter. Man fyllde i vissa svar och sen när man sparat det så skulle det komma en sammanfattning då. Så stämde det inte med vad man hade fyllt i. Sådant stör ju kundrelationen. Ligger ett sådant fel kvar för länge tröttnar ju kunderna och litar inte längre på systemet och så. Sådana grejer är ju viktigt att snabbt kunna åtgärda, tycker jag.

Och sen vissa andra grejer.. Till exempel om allting är rätt, men det dyker upp diagram som inte visar det det ska. Det tycker jag inte är lika bråttom, för alla uppgifter finns där och de stämmer. Det är bara liksom att se till att grafen visas på rätt sätt. Då tycker jag det är lättare att säga till kunden: ”Vi jobbar på det. Det är färdigt om en vecka”, liksom. Det viktiga är ju liksom.. Kunden måste ju få en känsla av att det är kvalitet och att det de fyller hanteras rätt och visas rätt. Sådana fel måste vi ta bort snabbt.

[F19] U: *Det här med koddokumentation. Ni har ju använt sådana här ’regions’ rätt flitigt. Är det någonting nytt eller det något som fanns sedan innan? Är det något ni tycker är viktigt?*

I: Jag vet ju bara hur Janne tänker.. Men han försöker ju lägga.. dela in det rätt så noga, vad metoderna är för sorter liksom. Om det är properties eller om det allmänna metoder eller privata metoder.

[F20] U: *Vad tjänar ni på det?*

I: Alltså, jag tror att han tänker att när han letar efter en metod så vet han ju i vilken region han ska leta i. Så blir det ju bara en femtedel så många att titta igenom. Sen är det ju också så att om det är någon annan som ska koda, så är det ju bra att veta var man ska leta.

[F21] U: *Hur är det då för den andra konsulten? Kommer han bara in och hjälper till lite då och då? Hur fungerar det?*

I: Janne är ju huvudansvarig över själva kodningen. Sen Fredrik, som han heter, hjälper Janne när det blir mer tekniska frågor. Mycket när det ska göras javascript, som ska liksom.. Till exempel javascript som kollar hur länge sedan man var aktiv. Där är Fredrik inne och sköter sådan kodning. De har någon sorts uppdelning sinsemellan, där Janne vet sin begränsning och då frågar han Fredrik, som har mer teknisk kunskap liksom. Vi har inte så mycket kontakt med Fredrik.

Förra sommaren när vi skulle få systemet stabilt var Fredrik mycket inblandad. För då var det mycket om det här jag sa.. Att folk kände att de blev utslängda från systemet. För systemet kände inte av.. Satt man och skrev i en textruta, vilket man kan göra ibland, man sitter och formulerar sig och så. Satt man och gjorde det i 20 minuter och tryckte spara så sa det bara ’Ping!’ utloggad. Så försvann allt man skrivit. Det var borta. Folk blev rätt irriterade på det.

[F22] U: *Hur gick ni tillväga för att hitta det? Vi har ju sett att ni inte har så mycket felhantering alls liksom.*

I: Ja, jo den diskussionen har jag också haft med Janne. Och han och Fredrik, som väl har kodat ihop rätt länge, de har själva bestämt sig för att inte ha en massa ’try and catch’. Därför att de resonerar som så att om det smäller så är det bättre att det smäller rejält. Så att folk börjar skrika. Så får man leta upp felet och se till att det inte blir fel. För har man en massa ’catch’ så att folk fortsätter att jobba i systemet trots att det inte fungerar, så kan det ta mycket längre tid att få reda på det.

[F23] U: *Men man kan ju lösa det genom att i alla fall visa vad det är som är fel. Är det inte svårt, i till exempel det fallet där de blev utkastade, att få reda på exakt vad det var som var fel?*

I: Det var ju inte ett fel liksom. Det var mer att systemet var byggt så. Så folk började bli sura över att de inte fick en chans att stanna kvar i systemet. Att serversidan inte märker att man sitter och skriver. Jag tror vi löste det så att vi byggde in ett script som känner av att man faktiskt sitter där och skriver.

[F24] U: *Inga andra problem som ni stött på? Eller kanske något ni lyckats lösa genom att skriva om koden kanske?*

I: Mer just det här generella att i och med att så mycket var hårdkodat innan så var det väldigt ovänligt för förändring. Då fick vi gå ut och sälja att det här finns. Nu måste ni använda det som är gjort. Nu när Janne bygger det mer så att man ska kunna modifiera det efter kundens önskemål. Då är det mycket lättare att gå ut och träffa kunder. Då säger kunden: ”Nä, men vi använder inte den här biten”. Nä men då ställer vi in det så att det inte dyker upp för kunden. Någon vill lägga in helt andra frågor. Då fixar vi det. Att gå ifrån den här hårdkodningen.. Att koden går in och frågar databasen hur det är inställt, den är rätt bra tycker jag.

Efter intervjun tackade vi så mycket för samtalet.

REFERENSER

Agrawal. A & Khan, R.A. (2009): Measuring the vulnerability of an object-oriented design, *Network Security*, 10 (2009), pp. 13-17.

Andersson, T. & Rexfelt A. (1999): *Projektarbete i en konsultorganisation: en utvärdering av Enators arbetssätt för projektstyrning, PPS.* Examensarbete: VXU/MASDA/IV/E/--9938—SE, Institutionen för matematik, statistik och datavetenskap, Växjö university.

Binkley A.B. & Schach S.R. (1996): A comparison of sixteen quality metrics for object-oriented design, *Information Processing Letters*, 58 (6), pp. 271-275.

Britton, C. & Doake, J. (2005): *A Student Guide to Object-Oriented Development*, pp. 75-116. Elsevier Ltd., Oxford.

Brooks, F. P. Jr. (1987): No silver bullet: essence and accidents of software engineering, *IEEE Computer*, pp. 10-19.

Budgen, D. (1995): 'Design models' from software design methods, *Design Studies*, 16 (3), pp. 293-325.

Chatzigeorgiou, A., Deligiannis, I., Tsantalis, D. (2008): An empirical study on students' ability to comprehend design patterns, *Computers & Education*, 51 (3), pp. 1007-1016.

Cote, I., Heisel, M., Wentzlaff, I. (2007): Pattern-Based Exploration of Design Alternatives for the Evolution of Software Aarchitectures, *International Journal of Cooperative Information Systems*, 16 (3-4), pp. 341-366.

Dasgupta, S. (1989): The structure of design processes, *Advances in Computers*, Yovits, M. C., Ed., Academic Press, pp. 1-67.

Dictionary.com (senast uppdaterad 2010): Design (elektronsik). Tillgänglig på http://dictionary.reference.com/browse/design (Åtkomstdatum: 2010-04-27).

Ghezzi, C., Jazayeri, M., & Mandrioli, D. (2003): *Fundamentals of software engineering* (2nd ed.). Prentice Hall.

Hsueh, N.-L., Kuo, J.-Y., Lin, C.-C. (2007): Object-oriented design: A goal-driven and pattern-based approach. *Software and Systems Modeling*, 8 (1) pp. 67-84.

Journal Digital AB: Om Journal Digital (elektronisk). Till gänglig på http://www.journaldigital.se/sv/omjournaldigital.html (Åtkomstdatum: 2010-04-07)

llighoven, H. (2005): *Object-Oriented Construction Handbook: Developing Application-Oriented Software with the Tools & Materials Approach.* Elsevier Inc., San Fransisco.

Jacobsen, D.I. (2002): *Vad, hur och varför? Om metodval i företagsekonomi och andra samhällsvetenskapliga ämnen.* Studentlitteratur, Lund

Lawson, B. (1980): *How Designers Think.* The Architectural Press Ltd., London.

Liao, H. (2009): Design of SaaS-Based Software Architecture. *International Conference on New Trends in Information and Service Science*, 2009, pp. 277-281.

Longman Dictionary (senast uppdaterad 2010): Design (elektronisk). Tillgänglig på http://www.ldoceonline.com/dictionary/design_1 (Åtkomstdatum: 2010-04-27).

Mansfield, R. (senast uppdaterad 2005): *Has OOP failed?* (elektronisk). Tillgänglig på http://www.4js.com/en/fichiers/b_genero/pourquoi/Has_OOP_Failed_Sept_2005.pdf (Åtkomstdatum: 2010-04-8).

Melissa, L., McGregor, D., McGregor, R. (2000): A Software Development Process for Small Projects, *IEEE Software*, 17 (5), pp. 96-101.

Nationalencyklopedin (senast uppdaterad 2010): Design (elektronisk). Tillgänglig på http://www.ne.se/design (Åtkomstdatum: 2010-04-27).

Robinson, J.A. (2004): *Software Design for Engineers and Scientists*, Elsevier, Oxford.

Selvarani, R., Wahida, B., Prasad, K. (2010): Quantifying the Design Quality of Object Oriented System: The metric based rules and heuristic, *National Conference on Advanced Software Engineering*, pp. 54-62.

Sen, A. (1997): The Role of Opportunism in the Software Design Reuse Process, *IEEE Transactions on Software Engineering*, 23 (7), pp. 418-436.

Skov M.B., Stage J. (2002): Designing interactive narrative systems: Is object-orientation useful?, *Computers and Graphics (Pergamon)*, 26 (1), pp. 57-66.

Soni, D., Ritu, S., Kumar, M. (2009): A Framework for Validation of Object-Oriented Design Metrics, *International Journal of Computer Science and Information Security*, pp. 6 (3), pp. 46-52.

U.S. Department of Homeland Security, Federal Emergency Management Agency (senast uppdaterad 2009): *Small Project* (elektronisk). Tillgänglig på http://www.fema.gov/government/grant/pa/glossary.shtm (Åtkomstdatum: 2010-04-14).

Voran, P. (2009): *Web Application Design Patterns*, Elsevier, Oxford.

Witt B., Baker T. & Merritt E. (1994): *Software Architecture and Design.* Van Nostrand Reinhold, New York.

Zhu, H. (2005): *Software Design Methodology: From Principles to Architectural Styles*. Elsevier, Oxford.