Examensarbete

HÖGSKOLAN I SKÖVDE

DESIGN PATTERN RECOMMENDER(DPR)

En utvärdering av ett DPR system

DESIGN PATTERN RECOMMENDER(DPR)

An evaluation of a DPR system

Examensarbete inom huvudområdet
Datavetenskap
Grundnivå 15 högskolepoäng
Vårtermin 2015

Mimmie Diits

Handledare: Mikael Thieme
Examinator: Joe Steinhauer

Sammanfattning

Recommender system blir allt vanligare och att kombinera recommender system med användningen av design patterns kan vara ett steg i rätt riktning för att komma närmre tekniker och program som skulle kunna öka kvaliteten hos programvaran i programvaruutveckling genom att öka utbyggbarheten och underhållbarheten hos programvaran.

Syftet med arbetet i den här rapporten är att utvärdera ett design pattern recommender program. Programmet kommer att utvärderas med hjälp av intervjuer av utvecklare och fallstudier hos tre företag. Frågorna i intervjun kommer att bygga på kriterier som används för utvärdering av CASE verktyg.

Nyckelord: Design pattern, case verktyg, utvärdering, programvaruutveckling

Innehållsförteckning

1 Introduktion

Den nuvarande stora utmaningen inom programvaruutveckling är samspelet mellan kvaliteten på programvaran, kostnaden och schema/tidsåtgång av projektet. Det kommer hela tiden nya verktyg/program för att hjälpa till med att förbättra kvaliteten hos programvaran det vill säga case tools/case verktyg (Daneva et al., 1996). Arbetet i den här rapporten kommer att rikta sig till att utvärdera ett case tool som är ett design pattern recommender program som hjälper utvecklaren att utveckla bättre programvara. Programmet hjälper till med att välja design patterns utifrån nyckelord som ingår i problembeskrivningen av programmet.

Design patterns kan liknas till en mall eller ett recept för hur koden i ett program ska skrivas för att den ska bli mer utbyggbar och lättare att underhålla. Design patterns används för att programmera så att vanligt förekommande fel som till exempel cirkulära beroenden mellan filer inte uppstår (Gamma et al., 1995). Programvaruutvecklingsprocessen utvecklas ständigt och ett gemensamt mål för alla utvecklare är att ha en effektiv process som i sin tur resulterar i en produkt eller system med kvalitet. Om det eftersträvas att gynna kvaliteten och utbyggbarheten hos programvaran väldigt tidigt i projektet kommer det att leda till att den kommer bli lättare att underhålla senare (Wohlin, 2005).

Utvecklingen går framåt inom design patterns och forskarna har gjort analyser (Aoyama, 2000) på hur design patterns påverkar programvara och i de fall där det inte handlat om kritisk prestanda så har det visat sig att det är en positiv inverkan att medvetet förbättra sin kod med för allmänheten kända medel. Eftersom att det finns ett stort antal olika design patterns så har forskarna kommit fram till att det är svårt att analysera dem var för sig och därför så har de delat in dem i pattern families (Aoyama, 2000). Pattern families är design patterns med samma syfte som har buntats ihop. Anledningen till att pattern families används är att det underlättar när man ska analysera design patterns. Eftersom att det har uppstått ett flertal olika design patterns kan det vara svårt att få en överblick över dem (Aoyama, 2000).

Det som egentligen är det smidigaste med design patterns är att det får utvecklare att programmera efter en viss standard och då är det lättare för alla inblandade att tolka koden och att kunna utveckla vidare på den. Genom att använda design patterns så blir programmeringen mer enhetlig samtidigt som den blir lättare att underhålla för att det är mer känt vad den gör och hur den är uppbyggd då den följer en mer standardiserad mall (Gamma et al., 1995).

Det upplevs som ganska svårt och krångligt att lära sig olika design patterns för utvecklarna då det finns en hel uppsjö av dem och som resultat så har många utvecklare i sitt arbetsliv eller utbildning hittat ett design pattern de känner sig bekväma med och som de sedan applicerar på all sin utveckling. Detta leder oftast till att de gärna använder just sitt favorit design pattern i sin programvaruutveckling även då det i vissa fall kanske skulle ha passat bättre med ett design pattern som är mer specifikt lämpat för uppgiften (Clark et al., 2000).

Det är lätt att bli lite hemmablind och hålla sig till det pattern som är känt för utvecklaren istället för att pröva ett pattern som kanske ger bättre resultat i slutändan. Ett nytt design pattern är kanske inte lika bekvämt att använda för utvecklaren eller så ligger det inte i utvecklarens intresse att lära sig nya saker. Det är lite det problem som en design pattern

recommender program är till för att lösa. Programmet hjälper utvecklaren att välja design pattern utifrån vad utvecklaren vet att programmet ska användas till.

Det finns många program som har hjälpfunktioner för programvaruutveckling. Några av de vanligast använda är:

- Microsoft Visual Studio 2012 – Utvecklat av Microsoft och har bland annat stöd för refactoring
- Eclipse Indigo – Utvecklad av Eclipse group och har stöd för olika språk och är gratis
- Xcode – Utvecklad av Apple för mac utveckling

Programmen har positiva sidor på olika sätt men de har inte stöd för att välja design pattern så därför behövs ett program med stöd för det. Med andra ord de är inte DPR program.

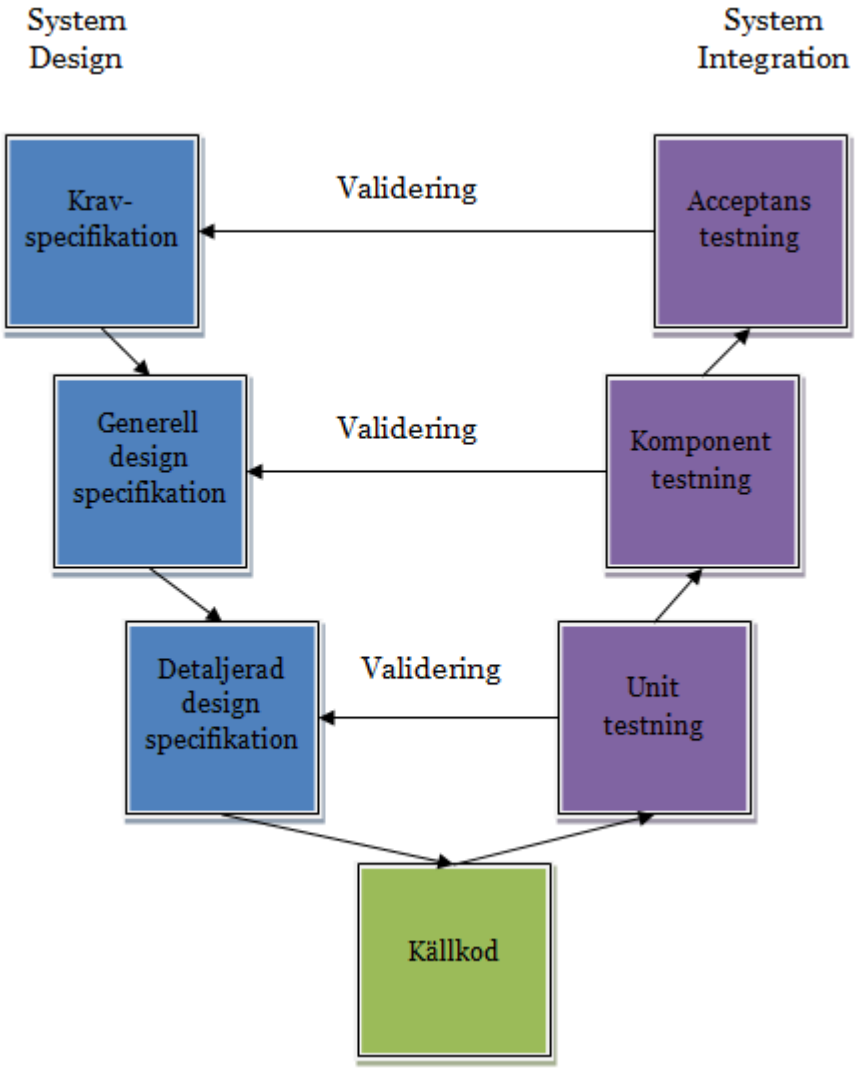
Ytterligare en metod som kan hjälpa till med att spara in tid och pengar vid programvaruutveckling är refactoring som är när gammal kod får sig ett ansiktslyft och förbättras utan att det blir någon ändring på vad programmet utför (se Kapitel 2.5). Det är likt design pattern recommender programmet ett sätt att förbättra sin programvara och öka utbyggbarheten och underhållbarheten.

2 Bakgrund

Syftet med arbetet är att utvärdera design pattern recommender som är ett program som hjälper till med att välja design patterns och i underkapitlen som följer så kommer alla nödvändiga delar för förståelse av ett sådant program och dess sammanhang att tas upp.

2.1 Programvaruutveckling

Programvaruutveckling är det sättet som programvara utvecklas på, oftast genom en utvecklingsprocess som brukar bestå av olika faser som oftast men inte alltid genomförs iterativt (Pressman, 2005). Med iterativt så menas det att faserna upprepas tills det resultat som önskas uppnås, data tillkommer efter varje iteration. Det finns olika modeller över hur en utvecklingsprocess kan se ut, en av dem är v-modellen (se Figur 1) som beskriver en agil process som oftast används i programvaruutvecklingsprojekt. Ordet agil härstammar från engelskans agile som betyder smidig, vig och lättrörlig. En agil process är ett sätt att jobba som är mera flexibel än en icke iterativ process. Det bygger på ett nära samarbete med kunden/användaren under hela utvecklingsprocessen (Abrahamsson et al., 2003). En agil process bygger på att den är iterativ. Ett annat alternativ är vattenfallsmodellen som bygger på att varje programutvecklingsfas skall vara färdig innan nästa påbörjas dock är inte den processen iterativ vilket gör att kostnaden för att rätta till kvalitetsproblem ökar med varje steg.



Figur 1 V- Modellen

Programvaruutveckling sker oftast i projektform. Ett projekt är mål orienterat och utförs inom en utsatt tidsram, med en budget samt består av ett antal aktiviteter t ex krav specifikation, design och implementation m.m. Först och främst måste problemet som projektet ska lösa definieras, sedan ska problembeskrivningen brytas ned i mer konkreta projektmål med delmål som kan kallas milstolpar i projektet (Pressman, 2005). En kravanalys görs och en kravspecifikation tags fram för att se till att utvecklarna och kunden har samma mål med projektet. En riskanalys är viktig att göra i ett projekt för att bedöma de risker som kan finnas som t ex brist på tid, brist på pengar eller andra olika typer av förseningar (Wohlin, 2005).

Det är viktigt att planera sitt projekt och hitta alla de fel som kan uppstå i projektet i ett tidigt skede för det är billigare att rätta till fel i början av ett projekt än att behöva göra om allt arbete i ett senare skede av projektet för då slösas massa värdefulla resurser. Projektet organiseras det vill säga att personer blir tilldelade att utföra aktiviteterna som tidigare bestämdes. De blir tilldelade aktiviteter utifrån den egna specialist kunskapen och lämpligheten (Wohlin, 2005).

Utvecklingsprocessen fungerar som ett ramverk för arbetet i projektet och är till för att personalen i projektet inte ska behöva fundera på vad som ska göras härnäst. Processen är till för att överbygga det avstånd i kunskap som kan uppstå mellan de olika personerna i personalen och ge stöd dem genom att ge struktur till arbetet och en klarare bild över vad som behövs göras och i vilken ordning (Wohlin, 2005).

I många organisationer har de en särskild avdelning av personal i projektet som hanterar kunder och marknaden för produkten/systemet ett så kallat kundteam. Viktigast är att utvecklarna har en god bild av marknadens eller kundens önskemål. Det är viktigt med kommunikation mellan kundteam och utvecklare.

En utvecklingsprocess innehåller även en designfas som är väldigt viktig i projektet för det är där som den första övergripande bilden av systemet/programmet skapas. Det går att modellera systemet för att se hur alla delar av det hänger ihop och testa sin modell för att hitta fel och brister så tidigt som möjligt (Wohlin, 2005). Det är i den här fasen som Design Pattern Recommender kommer att spela störst roll. Först så görs en mer generell design som sedan blir mer detaljerad och genomarbetad.

Efter designfasen kommer implementationen av systemet och då är det fördelaktigt att ha en bra design för att programmeringen ska gå så smidigt som möjligt. Om programmet har en bra design så kommer det att göra koden mer läsbar och lättare att förstå samt att det kommer att öka användbarheten av koden då det kommer att gå att lättare återanvända den och att bygga vidare på den. Programmeringen i implementationsdelen av projektet är oftast den första kontakten som relativt nya utvecklare får med programvaruutveckling. De får sin första introduktion i form av kurser som inriktar sig på programvaruutveckling som är väldigt inriktade på programmering eller av att de har suttit och programmerat hemma som hobby (Wohlin, 2005).

Det som är kvar att göra inom programvaruutveckling är att verifiera och validera programmet/systemet. Frågorna ”Är programvaran rätt?” det vill säga är den rent tekniskt korrekt och ”Är det rätt programvara?” det vill säga är det den programvaran som kunden/användaren var ute efter ska kunna besvaras i verifierings- och valideringsfasen (Wohlin, 2005). Om programvaran är felaktig så fungerar inte programmet och om det är fel

programvara så vill inte kunden/användaren ha programvaran och då har allt arbete varit i onödan. Unit testning och komponent testning görs för att kontrollera att programvaran gör som den ska göra. Sedan görs så kallad acceptans testning vilket är en form av validering och då görs det en avstämning med kunden att programvaran är enligt den kravspecifikation som tidigare tagits fram tillsammans med denne i början av projektet.

2.2 Programvarukvalitet

Kvalitet betyder olika saker beroende på typen av system. Säkerhet och korrekt exekvering är till exempel viktigare för system i kärnkraftverk än i datorspel. Ur utvecklares perspektiv är kvalitet mer att produkten är väl genomarbetad och så felfri som möjligt. Om en programvara är väl strukturerad så underlättar det framtida vidareutveckling och underhåll (Pressman, 2005).

Kvalitet kan delas in olika kategorier enligt ISO 9126 kvalitetsmodell (ISO/IEC TR 9126-4:2004) som har huvudkategorierna:

- Funktionalitet- där lämplighet, noggrannhet, samverkan, överensstämmelse och säkerhet ingår. *Dessa attribut indikerar hur väl programmet uppnår de specificerade kraven.*
- Tillförlitlighet- där mognad, feltolerans och återhämtning ingår. *Den mängden tid som programmet är tillgängligt att användas.*
- Användbarhet- där begriplighet, inlärning och handhavande ingår som attribut. *Svårighetsgraden att använda programmet.*
- Effektivitet- där tidsbeteende och resurs beteende ingår. *Den graden av optimal användning av systemresurser som programmet gör.*
- Underhållbarhet- där analyserbarhet, modifierbarhet, stabilitet och testbarhet ingår. *Hur lätt underhållet av programmet är.*
- Portabilitet- där anpassbarhet, installerbarhet, konformitet och utbytbarhet ingår. *Hur lätt man kan importera programmet från en miljö till en annan.*

Alla kategorierna och vad som ingår in dem är viktigt om man vill ha en programvara eller produkt av god kvalitet, sedan kan vissa av kategorierna prioriteras olika beroende på målet med programvaran (Pressman, 2005).

2.3 Underhåll av programvara

Det finns tre innebörder av underhåll och de är: anpassning, rättning och modifiering. Anpassning krävs för att till exempel möjliggöra mindre uppdateringar som kommer att behövas på grund av att kunna hantera en ny version av operativsystemet. Rättning innebär att de småfel som inte borde finnas i systemet har upptäckts och rättas till. Att modifiering krävs är på grund av att kraven kanske ändras och då måste programvaran modifieras (Wohlin, 2005).

2.4 Design Patterns

För att förstå design patterns måste först objektorienterad programmering gås igenom lite mer då design patterns appliceras inom just det området.

Objektorienterad programmering är ett sätt att programmera genom att uppfatta programmet som en modell av verkligheten som programmet ska samverka med (Skansholm, 2010). Begreppet objekt används för att beskriva dem enskilda enheterna i programmet och de är även dem modeller av något verkligt. Objektorienterad programmering är när färdiga komponenter som t ex klasser används ihop för att bygga ett program (Skansholm, 2010).

Ett begrepp som oftast används i objektorienterad programmering är det *semantiska gapet,* som är skillnaden mellan objekt och deras relationer i ett program och i verkligheten. I vissa fall finns det aspekter som inte går att ta med i programmet men som finns i verkligheten, dessa kallas semantiska förluster (Dorai et al., 2003).

Ett exempel på objektorienterad programmering är att skapa ett program över något som är välkänt och verklighetsförankrat som exempelvis en bil. Bilen ses som programmet och dess delar ses som objekt/klasser. Till exempel så har en bil hjul, en ratt, vindrutetorkare m.m. Alla dessa objekt tillsammans bildar det som i slutändan är en bil. Sedan så har objekten något som kallas attribut till exempel en ratt kan vara av läder eller plast och ha knappar som exempelvis volymknappar m.m. Attributen är klassens/objektets egenskaper.

Det handlar om att knyta programmeringen till verkliga relationer för att lättare få en överblick över programmet och för att få en större förståelse över hur det hänger ihop. Det låter programmeraren använda färdiga komponenter som sedan kan anpassas och kombineras ihop till ett färdigt program.

Design Patterns är generella lösningar på vanliga problem inom objektorienterad programmering. Dem är till för att kunna hjälpa till att lösa vanligt förekommande problem på ett väldigt generellt sätt för att kunna fungera för diverse olika programmeringsspråk. Design patterns är alltså sätt att programmera så att de här vanliga problemen inte uppstår. Exempel på vanliga problem kan vara att ett program behöver två klasser med ungefär samma egenskaper men att de skiljer sig åt bara litegrann. Problemet i detta är att nästan exakt samma kod kommer behöva skrivas i två olika klasser. Lösningen på problemet är att använda ett Decorator pattern. En Decorator klass skapas som innehåller endast skillnaden mellan klasserna och i och med det ”dekorerar” den andra klassen. Detta löser problemet att samma kod skrivs flera gånger (Gamma et al., 1995). Decorator förklaras lite bättre senare i detta kapitel. Ett ytterligare exempel på ett vanligt förekommande problem är om det är kritiskt att en klass endast ska ha en instans men att den även ska vara tillgänglig globalt och hur man uppnår det. Lösningen på det problemet är ett Singleton pattern. Detta pattern fungerar på så sätt att uppgiften att hantera om flera instanser av klassen skapas samt att den är globalt tillgänglig läggs på klassen själv (Gamma et al., 1995). Exempel på var detta problem kan dyka upp är i ett operativsystem där det endast bör vara ett filsystem.

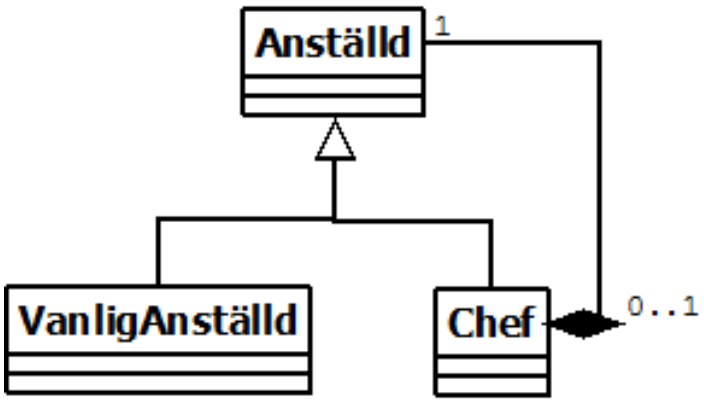
Design Pattern är oftast så pass generella att de är applicerbara i olika situationer men tillräckligt specifika för att hjälpa en att lösa problemen. De är mallar över olika sätt att programmera som hjälper en att undvika problemen (Palma et al., 2012).

Meningen med design patterns är att det ska bli lättare att underhålla koden och att den ska vara mer flexibel, alltså mer mottaglig för förändringar.

Enligt Gamma et al. (1995) så har ett design pattern fyra essentiella delar och de är:

- Pattern namn – Beskriver oftast problemet, lösningen eller olika konsekvenser som kan uppstå i ett ord eller två. Det är bra att namnge patterns då de blir lättare att diskutera med kolleger om de har namn men det är också väldigt svårt att hitta på bra förklarande namn.
- Problemet – Beskriver när det är lämpligt att applicera det pattern som valts. Det förklarar kontext för problemet. Ibland kan även problemet innehålla en lista av villkor som måste uppnås innan det ens är vettigt att använda sitt pattern.
- Lösningen – Beskriver de element som gör designen, relationerna, beroenden, och samarbeten. Lösningen beskriver inte en konkret implementation eller design utan är mer som en mall som kan användas i många olika situationer. Istället så är det mer en abstrakt beskrivning av ett design problem som kan lösas genom ett generellt arrangemang av element.
- Konsekvenserna – Är resultatet av att använda ett pattern. Det är konsekvenserna som styr användandet av patterns, de är kostnaderna de tar för att använda patterns.

I ett programvaruutvecklingsprojekt så bör design patterns användas av utvecklarna i designfasen och implementationsfasen av projektet där dem bör användas för att inte de vanliga problemen ska uppstå senare i projektet och för att säkerställa viss utbyggbarhet hos koden och att koden kommer följa en viss standard som bevisat ger mer kvalitet. Design patterns bör användas som mall för hur koden ska skrivas i projektet.



Figur 2 Decorator exempel

Ett exempel på ett vanligt design pattern är Decorator, det är ett pattern som ”dekorerar” det vill säga t ex ökar eller minskar ett befintligt värde på ett attribut. Om ett företag vill ha ett lönesystem med ett program som bestämmer anställdas lön kan det vara en god idé att ha en extra chefs klass då det ofta finns en löneskillnad mellan en chef och en vanlig anställd. Om nu löneskillnaden är det enda som skiljer chefen från den vanlige anställde så återkommer vi till problemet med att koden som är gemensam för chefen och den vanlige anställde kommer att behöva skrivas två gånger. För att lösa detta så kan man lägga den gemensamma koden i en ”Anställd” klass och låta klassen ”Chef” och klassen ”VanligAnställd” ärva den koden av ”Anställd” detta leder till att den information som klasserna behöver ha gemensamt inte behöver skrivas två gånger utan ärvs från ”Anställd” klassen vilket sparar in på rader kod som kommer behövas kompileras och sparar tid för programmeraren. För att i sin tur kunna lösa problemet med att chefen har ett annat löneattribut så läggs det till ett ”dekor” attribut i chefs klassen som ökar värdet på attributet ”lön” som klassen ”Chef” ärver från ”Anställ” klassen (Gamma et al., 1995). Detta är ett Decorator pattern exempel (se Figur 2).

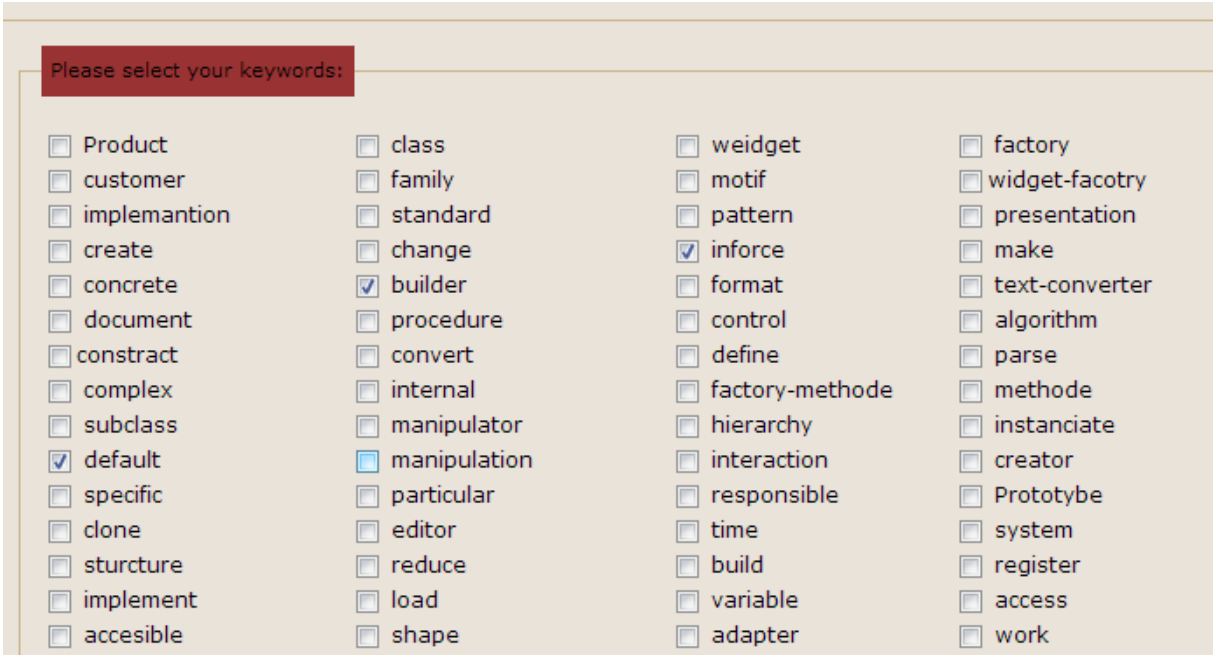
2.5 Refactoring

Refactoring är när redan existerande kod underhålls och om designas. Målet med Refactoring är att övergripande öka kodens utbyggningsbarhet och underhållet av koden, det är också ett av målen som man strävar efter i objektorienterad programmering när man tar hjälp av design patterns. Det som sker är att små förbättrande ändringar görs runt om i koden som sedan inte ändrar på vad koden gör utan bara på hur den gör det. Det är viktigt att regressionstesta, testa hela eller delar av programmet, för att vara säker på att omstruktureringen inte förstörde något (Yoder, 2011).

2.6 Design Pattern Recommender

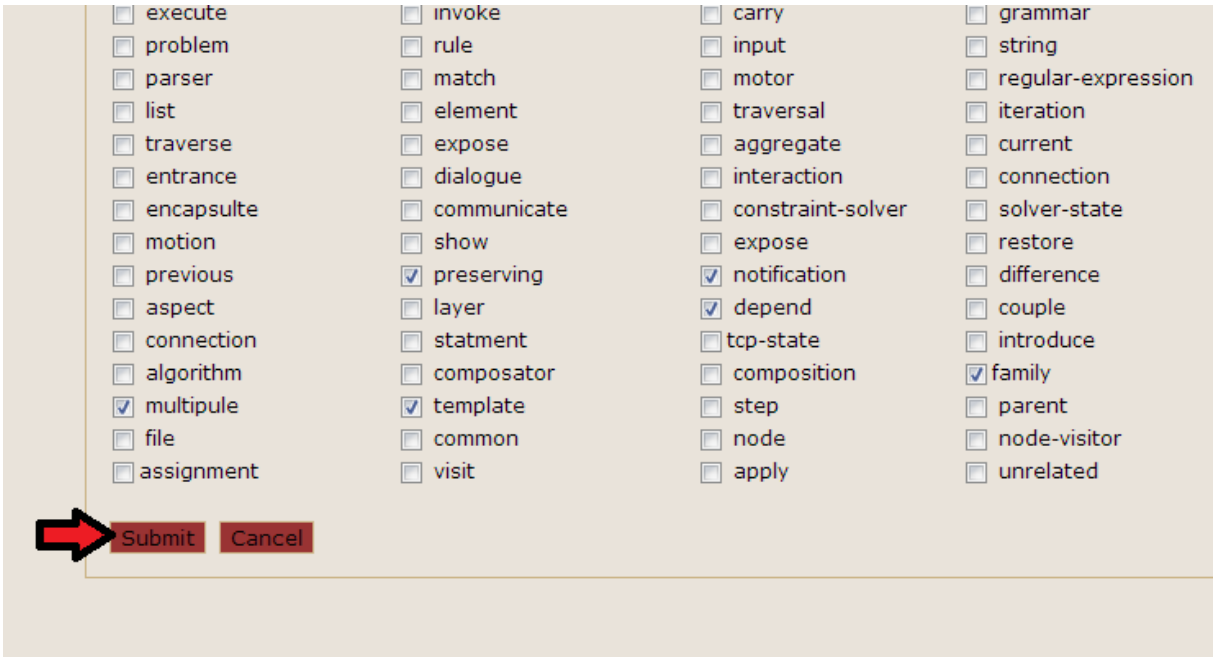
Recommender system används väldigt mycket inom e-handelförsäljning och används därigenom till exempel för att ge förslag på liknande varor eller varor andra användare som har köpt den varan man har markerad också har köpt. Det som spelar en viktig roll i ett recommender system är kvaliteten på rekommendationerna att på något sätt kunna rekommendera en vara som kunden skulle kunna intresseras att köpa till exempel (Manvi et al., 2011).

Design Pattern Recomender, DPR, är en programvara som är utvecklad av ett forskningsteam som kallar sig Ptidej (*Pattern Trace Identification, Detection, and Enhancement in Java*) och som med hjälp av vissa parametrar ska kunna hjälpa utvecklare att välja vilket design pattern som är lämpligast för att lösa deras problem (Palma et al., 2012).



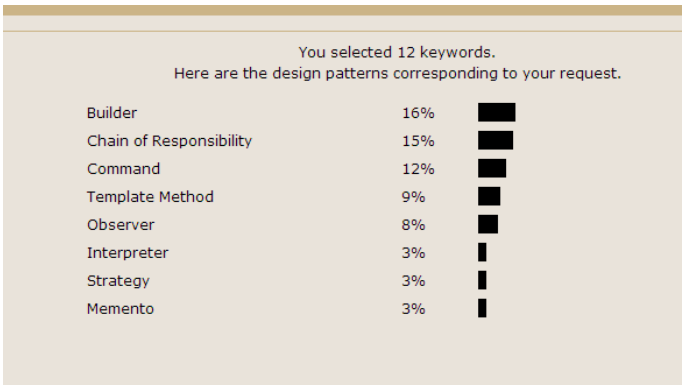
Figur 3 Lista på nyckelord

DPR programmet fungerar på så sätt att man klickar i alla de nyckelorden som ingår i ens problem så som det visas i Figur 3 och Figur 4.



Figur 4 Lista på nyckelord forts.

Sedan när nyckelorden har blivit i klickade så är nästa steg att trycka på submit knappen såsom det visas i Figur 4.



Figur 5 Resultat

Resultatet visas sedan i en lista av design patterns som är lämpligast för problemet i ordningen av det lämpligaste överst och sedan i fallande ordning (Se Figur 5).

Mer tekniskt så fungerar ett DPR program genom att design patterns väljs ut från ”*Design Patterns: Elements of Reusable Object-Oriented Software*” (Gamma et al., 1995) och sedan konstrueras en Goal-Question-Metric(GQM) modell för interaktiviteten, GQM är i princip ett mätinstrument för feedback och utvärdering. Som mål (Goal) så sätts pattern namnen och sedan på frågelagret (Question) så sätts två lager av frågor där det första lagret är villkor och andra lagret är undervillkor. Frågorna ställs till användarna av programmet genom att de får klicka i nyckelord (se Figur 3 och Figur 4). Varje nyckelord är sedan kopplat till de lagrade patterns och ger ett värde beroende på hur relevanta de är för olika patterns. Värdena räknas ihop och de patterns med högst värde alltså de mest relevanta blir rekommenderade. I det stora hela så går det ut på att beroende på vilka villkor som uppfylls genom att användaren gör sina val så får de design patterns som har flest kopplingar till dessa villkor högre procentuell lämplighets grad. Det pattern med flest kopplingar är det som är mest lämpat och

också det pattern som kommer att hamna högst av rekommendationerna (Palma et al., 2012).

Recommender system är designade att kunna fungera utefter användarnas behov och att vara så relevanta och ge så noggranna resultat på sina rekommendationer som möjligt (Jambor et al., 2012).

Det område som använder recommender system allra mest är tv/filmindustrin och sociala medier som till exempel Netflix, Facebook, Twitter etc. De använder sig av recommender system för att kunna rekommendera filmer, tv-serier och för att visa riktad reklam för användaren (Ciordas et al., 2010).

2.7 Utvärdering av CASE verktyg

CASE står för computer aided software engineering och är en benämning på verktyg som ska hjälpa till med olika delar av systemutveckling. Att utvärdera ett CASE verktyg bör ske efter kriterier som är fastställda och beprövade för att kunna lita på det resultat som utvärderingen ger enligt Senapathi (2005). Fokus är bäst att läggas på inlärningen av verktygen, hur snabbt eller lätt användaren förstår hur hon ska använda verktyget. Oftast är problemet med CASE verktyg att arbetsplatser och skolor anser att inlärningskurvan är för hög på dem och då används det inte på grund av att de inte vill lägga tid och pengar på något som dem inte ser någon vinst med (Senapathi, 2005). Därför är det viktigt att i utvärderingen av verktygen fokusera på inlärningen och att verktyget är värt att använda, underlättar för utvecklaren.

Lärbarhet i Case verktyg kan mätas genom tre kategorier: förtrogenhet, konsistens och förutsägbarhet (Senapathi, 2005).

- Förtrogenhet- Tidigare kunskap som skulle kunna hjälpa användaren att använda sig av Case verktyget.
- Konsistens – Att de olika funktionerna hos Case verktyget följer samma mönster så att användaren kan använda det med mer självsäkerhet efter att ha lärt sig grunderna, att programmet beter sig uniformt.
- Förutsägbarhet – Att användaren kan räkna ut resultatet av Case verktyget genom erfarenhet av tidigare användning av programmet.

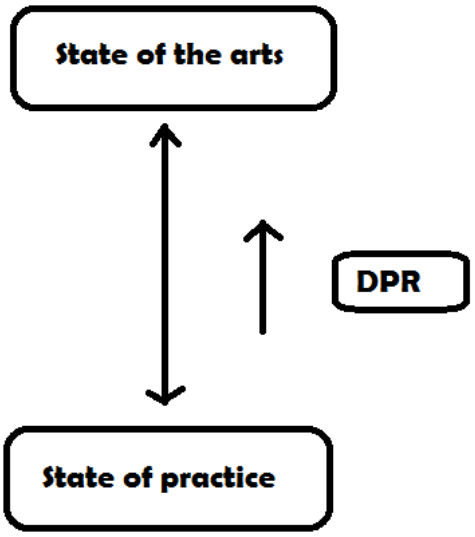
Generellt sett så är case verktyg till för att hjälpa oss med utvecklingen av programvara. Om ett verktyg har för hög inlärningskurva eller kräver allt för stor förkunskap så kommer dess användbarhet ute på företagen och i den akademiska världen att vara väldigt låg. Det ska löna sig att använda ett verktyg och kostnaden för inlärning får inte överstiga den nytta som verktyget kan göra för då är det ingen som vill använda verktyget (Senapathi, 2005).

3 Problem

Syftet med det här arbetet är att undersöka om DPR programmet är något som utvecklare skulle vilja använda för att hjälpa till att i sin programvaruutveckling förbättra kvaliteten hos programvaran de utvecklar och förbättra processen i deras programvaruutvecklings projekt. Då måste det undersökas hur lärbart och relevant programmet är. DPR gjordes 2012 och är i forskningsstadiet av ett kanadensiskt företag som heter Ptidej, det är nuvarande det enda programmet i sin genre (Palma et al., 2012) alltså recommender program för design patterns. En utvärdering av systemet bör göras för att det är ett intressant program med fördelar för programvaruutveckling och för att kunna bedöma värdet av att ha ett DPR program som hjälpmedel i utvecklingen av programvara. Recommender system har tillkommit mer och mer nu på senare tid då det har visat sig användbart i de olika delarna av utvecklingsområdet, främst inom området webbapplikationer som hanterar e-handel (Manvi et al., 2011). DPR är dock det enda recommender systemet för design patterns(Palma et al., 2012).

Om design patterns används i utvecklingen tidigt kan det senare minska den refactoring som senare bör ske, det hjälper till med att omorganisera programmet så att det går lättare att bygga på det. Så fort programvaran börjar användas bör den uppfylla två ibland motsägelsefulla krav, det första att uppfylla kraven i kravspecifikationen och det andra att vara mer utbyggbar (Gamma et al., 1995). Det går inte alltid att uppfylla alla kraven i kravspecifikationen samtidigt som programvaran är utbyggbar.

Det är svårt för företagen att ta åt sig alla nya program och idéer som kommer upp för att det är oftast en väldigt kostsam process att implementera nya system och få utvecklarna att bryta sina vanor och arbeta på helt nya sätt. En utvärdering av ett hjälpverktyg kan därför spara utvecklarna tid och pengar och det blir en extra validering av programmet där inte funktionaliteten står i fokus utan lärbarhet är av större vikt. Lärbarhet är en viktig faktor eftersom att företagen måste utbilda sina anställda i att använda programmet och då vill det helst inte att det ska ta för lång tid då det blir kostsamt för företaget om programmet är svårt att använda. Det påverkar även om företaget har beslutsfattare, som beslutar om programmen är värda att satsa på, sitter på en högre abstraktionsnivå inom företaget och inte är så insatta i vad utvecklarna skulle gynnas av för program. Det kan vara svårt för en icke insatt person att fatta ett sådant beslut. Det ska gynna företaget att använda programmet annars har det inget syfte. Utvärderingens syfte är också att undersöka om utvecklarna ser DPR som ett program som skulle vara relevant för dem.



Figur 6 Design pattern recommenders förmodade påverkan på utvecklingen

Med state of practice och state of the arts som visas i Figur 6 så menas det den nivå av utveckling som nåtts i forskningen av mjukvaruutveckling och den nivå av utveckling som faktiskt används i arbetslivet. Den stora pilen representerar gapet som finns mellan dem och den lilla pilen representerar den önskade effekten av DPR.

Varför en utvärdering av ett DPR program ens är intressant ur forskningsperspektiv är för att det blir mer och mer vanligt med recommender program för att hjälpa till i utvecklingsprocessen och mer vanligt i allmän programvara för e-handel (Manvi et al., 2011). Då är det av intresse att undersöka om recommender system är något utvecklarna är intresserade av och i det här faller DPR system.

Mål: Undersöka om användningen av DPR kan göra något för att minska avståndet mellan state of the arts och state of practice och om det finns ett generellt intresse av utvecklarna som ska ingå i undersökningen att använda programmet.

Det ingår även i undersökningen att undersöka vad de tre utvecklarna från de tre olika företagen som intervjuas skulle kunna se som en förbättring hos DPR programmet.

3.1 Metod

Det finns flera olika metoder för att kunna göra en undersökning till exempel litteratur analys, intervju, fallstudie, enkät, implementation och experiment (Berndtsson et al., 2008). De första fyra metoderna hade varit passande för undersökningen. Att endast använda litteratur analys var inte lämplig då den går ut på att systematiskt gå igenom och analysera utgivna källor utefter ett visst syfte och arbetet gick ut på att få feedback från användarna av DPR, enkät var inte heller lämplig då det oftast används för att undersöka välkända fenomen där det finns en stor samling av respondenter med god kunskap i området. Då det endast är tre personer med i undersökningen så blir en enkät inte så behjälplig då tre personer som inte har god kunskap i området inte passar in i kriteriet för en lyckad enkät som tidigare nämnts. DPR är ännu inte välkänt på marknaden och då fungerar det inte att göra en enkät samt att det är svårt att få tag på ett stort antal respondenter.

Den metod som kommer att användas i arbetet är en så kallad fallstudie. En fallstudie går ut på att ett program/system undersöks på djupet genom att till exempel intervjua olika relaterade personer om deras erfarenheter av programmet/systemet och sedan analysera den data man får in av undersökningen. En fallstudie per definition är ett sätt att empiriskt och detaljerat beskriva speciella aspekter av ett fenomen normalt baserat på flera datakällor (Yin, 2003). För att kunna undersöka om det finns ett generellt intresse hos utvecklarna att använda DPR är det lämpligt att ta sig ut på företag och intervjua dem. Därför valdes intervju som en metod i undersökningen.

Kvalitativ ansats, som är när fokus ligger på att resultatet av utvärderingen har hög kvalitet och inte på hur många resultat som ges, är den ansats som lämpar sig bäst i detta fall då det är ett system som ska utvärderas med hjälp av data som samlas in från utomstående. Därför är det lämpligt att göra en fallstudie och intervjua företag med öppna frågor som innebär att de ger flera svar istället för t ex ja/nej som sker vid slutna frågor.

3.1.1 Fallstudie

Fallstudier uppmuntrar till att använda olika metoder samtidigt. En fallstudie lämpar sig för att besvara frågorna hur? och varför? Fallstudier används för att sätta sig in i ämnet, öka förståelsen för det och att undersöka det på djupet. En fallstudie är när ett objekt väljs ut att undersökas och att subjekt används för att undersöka objektet (Berndtsson et al., 2008). I detta fall är objektet ett recommender system och subjekten är programvaruutvecklare som ska intervjuas för att kunna utvärdera systemet. En fallstudie går ut på att undersöka ett fenomen i dess naturliga miljö och ett karaktärsdrag hos en fallstudie är att det innehåller ett begränsat antal fall och att det ibland endast är ett fall som undersöks (Berndtsson et al., 2008). Fallstudier används oftast i områden som ännu inte är väl undersökta och välförstådda.

3.1.2 Intervju

Intervjuer kommer att samla in information om vad de tre utvecklarna har för kunskap om design patterns och vad de kan tänkas ha för nytta av programmet i sin utveckling av programvara och även vilka förbättringar som de skulle kunna komma på. Intervjuerna kommer att ske på plats på företagen och ska ge information om vad de tre utvecklarna anser om systemet. Det är en god idé att intervjua både före och efter observationen av användandet av programmet då de har olika synpunkter på användningen.

3.2 Tillvägagångssätt

Först så bestämdes det att en utvärdering av ett DPR program var en intressant sak att göra. Metoden som var lämpligast var en fallstudie med en intervju och en observation av användandet av programmet sedan så skulle det bestämmas hur många och vilka företag som skulle delta i intervjun. För att få en breddning av svar och för att ta hänsyn att undersökningen håller sig inom ramen för ett exjobb så blev beslutet att tre företag skulle delta och att en utvecklare från varje företag skulle intervjuas med frågor som var relevanta för att göra en utvärdering av programmet.

Utvecklare från tre olika företag kommer att intervjuas. Enligt Dumas et al. (1999) så ska man minst ha tre personer för en kvalitativ undersökning. Det kan även vara intressant att undersöka deras olika inställningar till ett DPR program.

För att kunna göra en utvärdering av programmet så krävs det en observation av utvecklaren som får testa programmet på plats för att kunna göra en rättvis bedömning av hur det skulle fungera i deras utvecklings miljö. Intervjuerna kommer också spela en stor roll då de hjälper till att ta reda på mer av utvecklarnas egna tankar om programmet och deras syn på hur det skulle kunna påverka effektiviteten och kvaliteten på just deras arbetsplats och om de skulle kunna tänka sig att använda ett sådant program i sin utveckling.

Anledningen till att göra en fallstudie är att det görs en undersökning av ett program i den miljö som det är meningen att det ska användas i. DPR programmet är byggt för att användas av utvecklare i en programvaruutvecklingsprocess och då är det lämpligt att observera en utvecklares användande av programmet.

Att ha en öppen intervju i detta sammanhang leder till tillgång av information som kanske inte skulle ha kommit fram om det istället hade frågats riktade frågor som inte lämnade rum för den intervjuade att komma med egna tankar om saker och ting. Det svåra med en öppen intervju är att om den intervjuande är oerfaren så kan det bli svårt att få intervjun att gå som planerat. Det är en stor risk då att intervjun går ifrån ämnet och då blir det svårt att styra tillbaka mot ämnet om intervjuaren inte känner till tekniker som skulle kunna hjälpa till att göra just detta. Intervjun kommer att spelas in och anteckningar kommer att föras undertiden för att komplettera. För att hindra att intervjun går ifrån ämnet så kommer förbestämda frågor ligga som en grund till intervjun och när intervjuaren märker att intervjun ändrar riktning kan intervjuaren använda sig av dem för att styra tillbaka i rätt riktning. Det blir en inblandning av struktur som kommer från en mer sluten intervju (Berndtsson et al., 2008).

Först börjas intervjun med förbestämda frågor som inte är formulerade för att bara ge ja/nej svar. Sedan får utvecklaren som intervjuas prova på att använda programmet och sedan avslutas undersökningen med en intervju efter observationen för att fånga upp utvecklarens tankar om att faktiskt ha använt programmet.

Så för att sammanfatta så kommer utvecklaren intervjuas innan användandet av programmet och sedan observeras under användandet av programmet för att tillslut få kompletterande frågor efteråt. Materialet kommer att spelas in och antecknas för att sedan transkriberas. Alla deltagare får information om hur intervjun kommer att gå till innan den görs. Resultatet kommer att analyseras i den här rapporten i ett efterföljande kapitel. Alla utvecklare som blir intervjuade får information om hur intervjun kommer att gå till i förväg, de är alltså medvetna i förväg om att den kommer att spelas in och transkriberas.

4 Genomförande

Då metoden bestod av en fallstudie och intervjuer så behövdes det utvecklare att intervjua så tre utvecklare från olika företag valdes, en från ett ganska litet företag med ca 5 anställda, en från ett medelstort företag med ca 20 anställda och en från ett stort företag med ca 10 000 anställda för att kunna få en inblick i hur de skulle använda programmet. De utvecklingsansvariga på varje företag intervjuades och fick testa programmet under observation. Vissa av dem var mer erfarna av att använda design patterns och andra hade inte någon större erfarenhet men eftersom det som mättes skulle vara hur ett design pattern recommender program skulle användas enligt just de olika förutsättningarna på företagen så spelade det inte någon roll om utvecklaren var erfaren. Det som krävdes var att de vet att design patterns finns så att de i alla fall kunde bilda sig någon form av uppfattning av hur ett sådant fungerar. För att sedan kunna bedöma om rätt design pattern hade valts av programmet behövs en viss kunskap om dem men det kan också vara en indikator på att de gör rätt i sin utveckling när programmet föreslår design patterns som de faktiskt känner till och använder i nuläget. De intervjuades både före och efter att de fick testa programmet för bästa bedömning av deras upplevelse av programmet. Intervjuerna skedde på plats hos företagen och spelades in på en bandspelarapp på telefonen.

Det var en utvecklare från varje företag som intervjuades och de fick svara på frågor om hur de arbetar, om de känner till design pattern recommender, om de tänker på utbyggbarhet när de utvecklar, om de skulle kunna använda programmet i sin verksamhet och hur inlärningen av programmet skulle fungera hos dem samt om de väl bestämde sig för att använda programmet om de skulle då använda sig av det befintliga programmet eller utveckla ett liknande själva. De intervjuade observerades när de testade programmet och fick svara på frågor om hur det var att använda programmet, om de fick det resultat de väntat sig, om tidigare kunskap av andra liknande program hjälpte dem med användningen av programmet och till sist om de skulle, efter att ha provat programmet, se någon användning av det i sin verksamhet. Frågorna och svaren finns i bilagor längst bak i rapporten.

Alla utvecklare som intervjuades hade erfarenhet av objektorienterad programmering och på så sätt kunde de sätta sig in i vad ett design pattern recommender program är till för. Företag 1 som är ett stort företag bidrog med en design arkitekt med insyn i den tekniska aspekten av mjukvaruutveckling, Företag 2 som är ett medelstort företag bidrog med sin lead programmerare och Företag 3 som är ett litet företag bidrog med en av sina programmerare.

De forskningsetiska principerna har ett individskyddskrav som i sin tur kan delas in i fyra olika huvudkrav: informationskravet, samtyckeskravet, konfidentialitetskravet och nyttjandekravet (Vetenskapsrådet, 2013).

De etiska aspekter som finns för själva programvaran: system quality, då det är viktigt att systemet är pålitligt och gör det som det är designat att göra för att främja utvecklingen av programvara genom användandet av design patterns. Slutanvändarens situation ses som utvecklarens ansvar. En annan aspekt är risk and reliability, då det är viktigt att man kan lita på att programvaran, i det här fallet DPR programmet, inte utgör någon risk för ens företag (Charlesworth et al., 2002). Då det inte används någon kritisk information från företaget och att programmet inte behöver installeras i deras utvecklingsmiljö så kan slutsatsen dras att det är tillräckligt pålitligt för att undersökningen ska kunna genomföras utan att programmet kommer att skada företaget.

Informationskravet enligt de forskningsetiska principerna har blivit uppnått då de intervjuade utvecklarna har fått information om hur intervjun ska gå till och på ett ungefär vilka frågor som kommer ställas under den. De har även blivit informerade om hur deras uppgifter kommer att behandlas. Alla har gett sitt samtycke att vara med och inga person uppgifter finns olovligen med så samtyckeskravet och konfidentialitetskravet är också uppfyllt. Alla uppgifter används i forsknings syfte så nyttjandekravet uppfylls också.

4.1 Utveckling av arbetet

Frågorna som först användes i intervjun var: Hur arbetar dem?, Tänker dem mycket på utbyggbarhet i sin designfas?, Känner dem till begreppet refactoring?, Skulle det hjälpa med ett DPR system i deras process? Kan de se fördelar? Nackdelar? Skulle dem kunna tänka sig att utveckla egna DPR system eller använda sig av befintliga? Syftet med dem var att de skulle hjälpa till att samla informationen som behövdes för att svara på frågeställningen i kapitel 3.

Det visade sig att resultatet från dem inte var så kopplat till problemdefinitionen utan kunde ses som irrelevant i relation till arbetet och vissa av dem kunde uppfattas som ledande som jag inte tänkt på att de kunde göras. Det som egentligen var syftet med arbetet. De var också aningen få och fick inte igång någon riktig diskussion och efter rådgörande med handledare så konstruerades nya frågor. Utvecklaren från företag 3 blev intervjuad först och blev sedan också intervjuad med de kompletterade frågorna och utvecklarna från företag 1 och 2 intervjuades efter det att frågorna blivit korrigerade.

Frågorna som användes för arbetet var:

Hur arbetar ni?, Tänker ni mycket på utbyggbarhet?, Har ni hört talas om design pattern recommender program förut?, Skulle ni kunna tänka er att använda befintliga DPR program eller utveckla egna?, Tror ni att tidigare kunskap av liknande program skulle vara till hjälp med inlärningen?, Tror ni att ett design pattern recommender program skulle fungera hos er?, Hur tror ni att inlärningen av programmet skulle fungera hos er?

Dessa frågor ställdes innan den intervjuade utvecklaren hade provat på programmet för att kunna hjälpa till och mata lite hur villiga de var att prova på nya saker i sin utveckling med andra ord ta ett steg närmare state of the arts från state of practice. Sedan när utvecklaren hade provat på programmet ställdes frågorna nedan som följdfrågor.

Hur var det att använda programmet?, Var svaret som programmet gav dig något du förväntat dig?, Kände du att din tidigare kunskap av liknande program hjälpte dig med just detta program?, Tror du att ni skulle kunna ha användning för programmet?, Finns det några förbättringar som du kan se?

Dessa frågor var mer kopplade till verklig utvärdering med anspelning på inlärning och förkunskapskrav och den intervjuades upplevelse av programmet och de var även mer öppna för diskussion. Syftet med frågorna är att samla in information som ger relevanta svar på frågeställningen i kapitel 3.

Att intervjun spelades in påverkade förmodligen hur de intervjuade reagerade på frågorna. Även om det skedde avskilt i ett rum med bara intervjuaren och den som blev intervjuad så sätter det ändå lite extra press på den som blir intervjuad om den är medveten om att allt som sägs kommer att spelas in.

5 Resultat analys

State of Practice (se Figur 6) just nu för alla företag som intervjuats är att alla jobbar i projekt enligt vad som är standard för företag som håller på med programvaruutveckling, ingen av utvecklarna som intervjuades hade hört talas om design pattern recommender innan utvärderingen men alla verkade positiva till programmet och användandet av det om flera förbättringar skedde med programmet. Utvecklarna som intervjuades är positiva till själva idén med att förbättra sin programvara med hjälp av design patterns. Det verkar även ligga en trend i att använda design patterns utan att vara speciellt medveten om det, t ex så var de intervjuade utvecklarna osäkra på exakt vad design patterns var men sedan när de hade testat programmet så kunde de känna igen några av de exempel på design patterns som dök upp i resultaten. State of Practice kan enkelt förklaras med det sätt som företagen jobbar på i nuläget, den nivå av teknologi och avancering som deras utveckling ligger på.

State of the arts (se Figur 6) för området design pattern recommender program är att det finns bara ett program hittills som är ett design pattern recommender program, det finns inget som säger att denna idé inte kommer att utvecklas mer och att programmet inte kommer att utvecklas mer för att få lite ökad användbarhet på arbetsmarknaden. Det är även mycket fokus på att analysera design patterns för att kunna optimera användandet av dem i utvecklingen av programvara (Aoyama, 2000). För de företag som intervjuades så är state of the arts ett mer aktivt användande av design patterns för att få ut så utbyggbar och underhållbar programvara som möjligt

DPR programmet har en möjlighet, baserat på svaren från alla intervjuerna, att vara till en hjälp med att minska gapet mellan state of the arts och state of practice. De intervjuade utvecklarna verkade positivt inställda till att använda design patterns för att förbättra sin programvara. Dock endast om vissa förbättringar av design pattern recommender programmet gjordes såsom att lägga till länkade svar till resultaten. Så själva principidén med DPR är god men att programmet i nuläget behöver förbättras innan det blir användbart.

Intervjufrågorna gav tillräckligt med information för att kunna utföra utvärderingen av DPR programmet och det gjorde även de tre utvecklare från de tre olika företag som valdes ut att delta i utvärderingen. All bakgrundsinformation i kapitel 2 kunde enkelt hittas på de databaser som finns länkade på högskolebibliotekets sida på his.se och i biblioteket självt.

Det som har varit genomgående gemensamt för alla utvecklare som blivit intervjuade är att alla har de kommit på förbättringar som kan göras på programmet. De tycker också att programmet är en intressant idé och verkar se ett verkligt värde med dess existens. Det som de tyckte saknades hos programmet var en motivering av resultatet, alla tre påpekade att det skulle vara bra att veta vad de olika patterns gjorde när man väl fick dem rekommenderade till sig. Att göra resultatnamnen till länkar med motiveringar var en idé som föreslogs av Företag 1. De tyckte att det hade varit mer användbart om en motivering på varför just det valda pattern var det bästa för de nyckelord som dem hade valt.

För att komplettera intervjuerna så observerades användandet av programmet och följdfrågor ställdes efteråt för att få ett så täckande resultat som var möjligt. Utvärdering av case verktyg, som är en kategori som man kan räkna DPR programmet till, är mycket inriktat på lärbarhet för det är vad som oftast är den mest övervägande kvaliteten som räknas in när man beräknar om det är värt att börja använda ett verktyg eller inte. Utvecklaren från företag

1 tyckte att lärbarheten var viktig då hen upplevde att kostnaden av att utbilda personal spelade roll i införskaffandet av verktyget.

> *”Jag menar om vi skulle säga att vi har 1000 användare av applikationen och vi har 2 timmars upplärnings tid är det 2000 timmar och 2000 timmar gånger timpenningen är typ 2miljoner eller vad det kan vara. Det är en stor del att det kan vara absolut enkelt att använda, det är otroligt viktigt så själva poängen där är att användningen är viktig eftersom att det är en kostnad.*”
> – Design arkitekt Företag 1

Alla utvecklarna verkade positivt inställda till att DPR programmet skulle kunna förbättra kvaliteten på deras programvara och tyckte även att det skulle få dem att börja använda design patterns mer som i sin tur skulle öka utbyggbarheten hos deras programvara om DPR hade genomgått förbättringarna som tidigare nämnts. Programmeraren från Företag 3 nämnde: ” *Det hade nog funkat till en viss grad. Om det hade varit vettigt, om man kan lita på det. Vi använder inte så mycket design patterns så det kanske är en bra start. Om man får lite tips.*” när frågan om programmet hade fungerat i deras verksamhet ställdes.

Utbyggbarhet var en punkt som alla utvecklarna nämnde som en viktig punkt i sin programvaruutveckling.

Lead programmeraren på Företag 2 svarade: ” *Ja det gör man väll absolut. Det är väl en av dem sakerna man gör fel på kanske när man programmerar att man kodar lite för generellt, att man ska kunna bygga ut på det även om man kanske inte behöver det. Men det blir väll så det är roligare att koda på det sättet, man bygger nått generellt man kanske har nytta av nån gång i framtiden.*” på frågan om de tänkte på utbyggbarhet när de utvecklar.

Utvecklarna gav också en inblick i hur de såg på inlärning av nya program och hur de skulle gå tillväga för att utbilda sin personal i att använda ett design pattern recommender program. Utvecklaren från Företag 1 beräknade sig få en större kostnad på grund av att mängden anställda som skulle behöva utbildas skulle bli väldigt mycket större än för de små företagen vilket i längden inte skulle göra det lönsamt för dem.

På frågan om de trodde att tidigare kunskap hjälpte dem att använda programmet så svarade samtliga intervjuade att ja det trodde dem, men programmet var väldigt enkelt att använda. Det svåra som de verkade komma överens om var att det krävdes viss förkunskap om design patterns för att kunna få ut något vettigt från resultatet som programmet gav dem.

Det verkade även att utvecklaren från Företag 1 som är ett stort företag inte hade någon lust att utveckla något eget liknande program till DPR programmet utan ville hellre använda sig av det befintliga programmet. Utvecklarna från de mindre företagen var mer intresserade av att utveckla eget om de bara fick tid på sig att sätta sig in i programmet.

Utvecklaren från företag 1s tolkning av varför DPR programmet inte skulle fungera hos just dem var att när det väl skulle behöva läggas in i utvecklingsprocessen så skulle de som skulle behövt modellera det i ett tidigt skede ha bristande så kunskap om just den tekniska biten att de skulle vara omöjligt för dem att förstå det design patterns som programmet skulle föreslå. Allt detta för att dem har så stor organisation och att de får sina krav och riktlinjer från en

ganska hög nivå inom organisationen och dem i sin tur kanske inte alls har koll på den tekniska biten.

Då de tre företagen inte använde sig så aktivt av design pattern i sin utveckling gick det inte riktigt svara på om det design patterns som de blev rekommenderade var några som de själva skulle ha valt.

Metoderna intervju och fallstudie påverkade den data som samlades in genom att det förmodligen fick utvecklarna att bli lite nervösa av att intervjun spelades in och det kan ha påverkat resultaten då de kan ha svarat utefter vad de trodde var förväntat av dem. Dock så var en fallstudie ytterst lämplig då arbetet gick ut på att fördjupa sig inom ett specifikt område av programvaruutveckling som i princip är det som en fallstudie går ut på, att man fördjupar sig på ett fenomen inom ett område (Berndtsson et al., 2008). För att försöka motverka nervositeten så började intervju med en kort presentation av arbetet och mig som intervjuare för att de skulle känna sig mer bekväma med situationen. Det verkade inte heller vara någon av de tre utvecklarna som reagerade på att intervjun spelades in. För att de skulle känna sig mer bekväma med inspelningen så var det viktigt att inte göra ett stort spektakel av det utan bara lugnt och fint sätta igång inspelningen då de redan hade blivit tillfrågade om det var okay att intervjun spelades in. Då kunde de fokusera mer på frågorna och bete sig mer naturligt.

Att endast tre utvecklare intervjuades påverkar såklart resultatet i den mån att det blir begränsat till tre individers åsikter även om det är inom ramen för en kvalitativ undersökning att intervjua minst tre personer (Dumas et al., 1999). Resultatet skulle säkert skilja sig om antalet intervjuade var fler men resultatet i nuläget är tillräckligt inom ramen för arbetet. Det finns inte heller något med undersökningen som påverkar företagen som utvecklarna arbetar på så det fanns heller ingen anledning för utvecklarna som intervjuades att inte svara ärligt på intervju frågorna.

Det finns två stora hot mot validiteten hos arbetet när det kommer till kvalitativa undersökningar. Den första är att man misslyckas med att ta hänsyn till att beräkna sina egna fördomar och partiskhet och låta dem influera arbetet med undersökningen (Berndtsson et al., 2008). Till exempel i detta fall att själv ha god kunskap om design patterns och sedan förvänta sig att andra utvecklare också skulle besitta den kunskapen. För att kunna frångå den risken var det viktigt att redan från början inte ha god kännedom som ett krav på de utvecklare som skulle intervjuas. Det andra hotet mot validiteten är ens egen påverkan på undersökningen till exempel att inte visa de som blir intervjuade transkriberingen av intervjun och att sedan basera påståenden i analysen på materialet från intervjuerna (Berndtsson et al., 2008). För att motverka detta så har de transkriberade intervjuerna lagts som bilagor i rapporten.

Det som framkommit genom arbetet är att det finns ett intresse hos företagen av ett recommender verktyg för design patterns. De lite mindre företagen (Företag 2 och Företag 3) tycker att det verkar intressant att själva vara med och vidareutveckla liknande program för att kunna få bättre kvalitet och utbyggbarhet hos sin programvara.

6 Slutsats

Slutsatsen som kan dras om den här typen av CASE verktyg i ett större perspektiv och mer generellt.

Även om utvecklarna i nuläget inte tänkte så mycket på att använda design patterns överhuvudtaget så är de väldigt positivt inställda till ett verktyg som hjälper dem välja. Så med andra ord minska gapet mellan state of the arts och state of practice för dem.

Eftersom att målet med arbetet var att göra en utvärdering av design pattern recommender programmet och detta genomfördes med hjälp av intervjuer med frågor som byggde på kriterier för att utvärdera CASE tools (Senapathi, 2005) med inriktning på lärbarhet och användbarhet så kan slutsatsen dras att arbetet har gett ett positivt resultat och att huvudmålet är uppfyllt.

6.1 Sammanfattning av resultat

Design pattern recommender programmet togs emot på ett varierat sätt av de tre olika utvecklarna.

Utvecklaren på företag 1 som är ett stort företag såg inte att programmet i dess nuvarande form skulle kunna gynna deras verksamhet så det krävs en del förkunskap av design patterns för att kunna använda den output som programmet ger en. Utvecklaren tyckte även att det skulle bli för dyrt att utbilda de ca 1000 personer som i så fall skulle använda programmet med, hade det varit mindre beroende av förkunskap så skulle de kunna tänka sig att köpa in befintliga program.

Utvecklaren på företag 2 som är ett medelstort företag trodde att det skulle vara intressant att prova på att ha programmet och de såg inget problem med inlärningen av det. De var även öppna för att eventuellt utveckla egna recommender program och på så sätt bidra till vidareutvecklingen av dem.

Utvecklaren på företag 3 som är ett litet företag hade heller inget emot tanken att ha DPR programmet som verktyg i sin utveckling och var även dem öppna för att utveckla egna sådana program.

6.2 Diskussion

Den samhälleliga nyttan kommer att vara att få bättre kvalitet på programvara om företagen skulle börja använda detta verktyg som hjälper dem att välja design patterns som i sin tur leder till att utbyggbarheten ökar och att underhållbarheten blir lättare vilket leder till bättre och flexiblare programvara.

Även om programmet fortfarande ligger på forskningsnivå så finns det fler goda utvecklingsmöjligheter för det redan nu, som alla de tre utvecklarna verkade kunna bidra med idéer för.

Mer tid skulle kunna ha lagts på att kolla upp vilka företag som aktivt använder sig av design patterns i sin utveckling för att kunna få lite bättre resultat. Det som spelade in var att hålla sig inom ramen för ett examensarbete och då fanns det inte tid att göra en sådan

förundersökning och hitta högt kvalificerade företag i närheten och som ville ställa upp på en intervju. De svar som fallstudien och intervjun gav var tillräckliga för arbetet.

6.3 Framtida arbete

Det skulle nog gå att ta arbetet ett snäpp längre och implementera det i företagens verksamhet för att undersöka dess verkliga påverkan på kvaliteten hos deras programvara. Det skulle vara en intressant aspekt att jämföra resultatet av den undersökningen för att kunna dra någon slutsats om att företagets storlek spelar in på hur pass bra som programmet ökar kvaliteten. Det kanske även också skulle vara en idé att implementera de förändringarna som företagen föreslog för att ytterligare öka användbarheten hos programmet.

Även om framtida arbete i just DPR system inte ökar drastiskt så börjar recommender system överlag bli mer och mer intressant för forskningsvärlden och det undersöks hur man kan utvärdera dem och hur man kan använda dem inom olika områden (Manvi et al., 2011).

Referenser

Abrahamsson,P.; Ronkainen, J.; Siponen, M. T.; Warsta, J. (2003) *"New directions on agile methods:a comparative analysis"* Proceedings of the 25th International Conference on Software Engineering s. 244-254

Aoyama,M. (2000) *"Evolutionary Patterns of Design and Design Patterns"* Principles of Software Evolution, 2000. Proceedings. International Symposium

Berndtsson,M.; Hansson,J.; Olsson,B.; Lundell,B. (2008) *"Thesis Projects – A Guide for Students in Computer Science and Information Systems "* second ed. Springer

Charlesworth,M.; Sewry, D. A. (2002). "*Ethical Issues in Enabling Information Technologies". SAICSIT*. 1 (1), 163 - 171.

Ciordas,C.; Doumen,J (2010) *"An Evaluation Framework for Content Recommender Systems-The Industry Perspective"* 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 273-277.

Clark,D.; Della, L. (2000). *"Teaching Object-Oriented Development with Emphasis on Pattern Application*" Proceeding ACSE'00 Proceedings of the Australian conference om Computing education s. 56-63.

Daneva,M.; Terzieva, R. (1996). *"Assessing the Potentials of CASE-Tools in Software Process Improvement: A Benchmarking Study"* Assessment of Software Tools, 1996., Proceedings of the Fourth International Symposium on 22-24 May 1996 s. 104-108

Dorai C, Venkatesh S. (2003). *Bridging the Semantic Gap with Computational Media Aesthetics. IEEE MultiMedia 2003;10 (2):15-17*

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. (1995) *"Design Patterns: Elements of Reusable Object-Oriented Software*" 1st ed. USA: Addison-Wesley

ISO/IEC TR 9126-4:2004(en) *"Software engineering — Product quality — Part 4: Quality in use metrics"* ISO/IEC JTC 1/SC 7 Committee, 1st edition , 2004-04-01, ICS 35.080

Jambor, T.; Lathia, N.; Wang, J. (2012) *"Using control theory for stable and efficient recommender systems"* WWW'12 Proceedings of the 21st international conference on World Wide Web s. 11-20

Manvi S.S, Nalini. N, Lokesh. B. Bhajantri (2011)*" Recommender System in Ubiquitous Commerce"* Electronics Computer Technology (ICECT), 2011 3rd International Conference on (Volume:4) s. 434 - 438

Palma,F.; Farzin,H.; Guéhéneuc, Y-G.; Moha, N. (2012)" *Recommendation System for Design Patterns in Software Development: An DPR Overview" R*ecommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop s. 1-5

Pressman, R. (2005) "*Software engineering: a practitioners approach*" sixth edition. Singapore: McGraw-Hill

Senapathi, M (2005)"*A Framework for the Evaluation of CASE Tool Learnability in Educational Environments* " Journal of Information Technology Education, Auckland. S.61-70

Skansholm, J. (2010). "*Java direkt med swing*" Upplaga 6:1 Lund:studentlitteratur

Stroustrup, B. (2008)"*Programming, Principles and Practice Using C++*" 1st ed. Boston: Addison-Wesley

Vetenskapsrådet (2013) "*Forskningsetiska principer inom humanistisk-samhällsvetenskaplig forskning*" ISBN:91-7307-008-4

Wohlin,C. (2005). "*Introduktion till programvaruutveckling*". 1st ed. Lund: studentlitteratur

Yin, R (2003)"*Applications of Case Study Research*"**,** Second Edition: SAGE

Yoder, J.W. (2011)"*Refactoring at the Core of Agile Software Development*" The Refactory, Inc. AOSD '11 Proceedings of the tenth international conference on Aspect-oriented software development companion, Pages 51-52

Figur 1, V- Modellen *BCS ISEB Foundation Certificate in Software Testing - Syllabus* (NB PDF download). Accessed 9th September 2015 http://www.bcs.org/upload/pdf/ct-foundation-syllabus.pdf

Appendix A - Intervju 1

Företag 1, Design arkitekt

Hur arbetar ni?

Jo, bara lite kort hur vi arbetar egentligen alltså. [redacted] är ju ett stort företag vi är ju över 10 000 personer inom [redacted] och i o med det har vi ju också många regler för hur både utveckling ska se ut hur mjukvarudelen ska se ut och vi har ju väldigt många typer av roller inom organisationen och vi har också då ska vi säga ett centralt stöd som kommer ifrån en central organisation som skickar ut guidances på hur vi utvecklar GUI på hur projekten ska arbetas hur vi gör projekten. Vilka saker som är recuired om man säger så och vilka saker som är nästan optional och så vidare, precis det är väldigt inriktat på mer att försöka ha en business funktionalitet och att se till att den verkligen blir rätt. Det är mycket mindre inriktat på dem patterns eller design p patterns som utvecklare använder sig av för att uppnå målet som då ligger på business. Och därför tycker jag att det ska bli spännande att se vart nånstans i gränslandet som den här applikationen kan ligga nånstans därför att idag så är det så att som jag sa att det pratas och det är väldigt mycket business och det är egentligen bara i sista skedet som när man omvandlar de här businesskraven in mot applikationen för att stödja de funktionaliteter som behövs och det är ju ett problemområde om vi ska uttrycka det så idag därför att där krävs det så många mellanhänder när man går från en businessfunktionalitet in till dels en vart saker ska gå in i applikationen alltså vilka delar och vilken typ utav lösning de bör vara och sen har du, och det är egentligen arkitektens roll eller lead arkitekten som har att se till att den förs in. Och sen så har du en annan typ och det är då själva utvecklingen av just den här arkitekturen som arkitekten tagit fram som då ska implementeras och sen går de tillbaka som ett varv och tittar på har det verkligen gått in så att den täcker alla businesskrav vi har. Men i alla fall det är en fråga som hela tiden förändras och det arbetssättet förändras och alla grupper tror jag arbetar lite olika runt om just det här beroende på hur komplex och stor applikationen är. Som det är just idag så arbetar jag i en grupp där teamet för ett fabrikskontrollsystem över 14 personer och då inkluderar inte det business alltså det som ligger ute de som bestämmer på en lite högre nivå utan det här är enbart för det programmet och den är en del utav ett mycket större paket som inkluderar många, många fler applikationer då som ska rullas ut samtidigt då på olika sajter vi har runt om i världen för tillverkning utav lastbilar. Så det är lite kort om hur det är.

#förklarar för den intervjuade hur intervjun kommer gå till om att en del frågor ställs innan testen av programmet och en del ställs efter och ger en förklaring hur programmet fungerar rent tekniskt.

Det ska bli intressant att se huruvida det matchar det tänkta inmatnings mönstret gentemot vad man faktiskt har ute på en verksamhet. Det är det som jag är intresserad av, det är det som alltid är ett problem det här med att när man tittar på det som finns i den perfekta världen så säger den akademiska delen att det är det här som liksom om man använder det här så blir allt bra, men det är oftast så att när du väl kommer ut till en organisation så är det inte att man har det här så därför så blir det intressant och se nånstans var om det finns nån översättningsmöjlighet mellan det som faktiskt finns i dokumentation över till det som krävs eller om det direkt går in eller om man helt enkelt ser det som två olika saker. Så det är som sagt det tycker jag ska bli intressant att se

 # lite syftet med programmet tas upp

För övrigt kan jag bara nämna då att vi, det här ADT Application Development Technique dom har olika avdelningar inom java .net mainframe och alltihopa som har tagit fram de här standarderna så de e så att dom har ju tagit fram en arkitektur med sample applikation som är baserat på domain driven design det av hierarky som står som pappa till det och sen byggt vidare på den och då koppla upp den mot repostor pattern som vi alltid använder i bakrunden mot alla våra data källor. Vi använder till exempel controller på mvc på web applikationerna. Vi ser till så att vi har en viss typ som vi alltid följer mellan dom olika lagerna i vår applikation och så vidare. Så där det är så man utgår från varje applikation som vi gör. Däremot så blir det vissa delar som inte, dom här patterns som kommer in och lägger sig lite grann emellan för att lösa vissa typer av problem men det här är just det ja sa förut dom här problemen är oftast inte nånting som artikuleras därför att de tillhör inte business.

Har ni hört talas om design pattern recommender program förut?

Jag har personligen inte tittat på design pattern recommendation men däremot så har jag både studerat och läst litegrann angående indexerings applikation eller indexeringssystem för att titta på den generella säkerheten i applikationer men inte direkta för design patterns det har jag inte gjort.

Tror ni att ett design pattern recommender program skulle fungera hos er?

Jag tror att, om det är kopplat till ett direkt förslag på hur implementationen skulle se ut, det hade inte gett så mycket om det bara kommer fram ett namn på ett design pattern att oj det här skulle se bra ut. Däremot om det hade varit kopplat till, till exempel förslag för en standard implementation hur den skulle i grova drag sett ut med de här design patterns för att visualisera för användaren varför det gör så. Där skulle jag kunna se en användning men jag har svårt att se hur det skulle kunna gå in därför att man tar som du säger det man redan vet. Och är det så att det bara kommer fram ett namn på nånting så kan det vara svårt.

#

Ju större företag när man har de här .. så är det så att det finns väldigt många olika nivåer och roller på saker du gör och det är det här som är att den som ska göra jobbet inte har ansvaret att dra upp arkitekturen alltid så då blir det liksom att det är liksom försent när själva utvecklaren ska göra det här för då måste det vara arkitekten som gör det och arkitekten har inte gått ned på den tekniska delen än så det är liksom att det blir hönan och ägget generellt på ett större utvecklings team. Det är ju som sagt att organisationen kan förändras om man ändrar om att man lägger in det här som gemensamma uppdrag att ta fram en, titta på alla krav som finns och försöka få in de nånstans i applikationerna och titta på därifrån om man kan se en.. samtidigt är det så att oftast , det är inte så många gånger som vi faktiskt gör nya applikationer . mycket utav det man gör är vidare utveckling på saker som redan finns och oftast är det så att vi har ju våra tidsspann man börjar med att vi får ut en release till exempel och efter det att grunden finns så är det bara på med nya features hela tiden. Och varje sån här feature brukar nästan bara vara plugga in i det läget likadant som dom andra featuresarna har gjort det är bara att vi har en annan funktionalitet so har naturligtvis. Så fördelar: hade vi fått in den i en bra rytm och man kunde se/lära sig utav den då hade jag kunnat se de som en fördel. Nackdelen är naturligtvis att vi jobbar ju agilt inom Företag 1 och om det är så att det inte ger nånting så är det som waste och försvinner ut

direkt asså det här att det får inte ta för lång tid att sätta sig in i man måste kunna lita på resultat eller att det ger nånting helt enkelt.

Skulle ni kunna tänka er att använda befintliga DPR program eller utveckla egna?

Nä det är befintliga. Vi gör ju mycket själv men det vi gör själv ska vara för en specifik uppgift som är kopplad till vår organisation,. Vi tillhör ju [redacted] alltså ska vår kärnverksamhet vara för att utveckla IT inom [redacted], koncernen [redacted], och vi ska inte lägga vår kunskap på i så fall nått annat än faktiskt det som driver [redacted] framåt. Så i de här lägena är det absolut billigast och bäst att köpa in systemet precis som vi gör med alla de andra systemen.

Hur tror ni att inlärningen av programmet skulle fungera hos er?

Ja det är ju rätt så intressant. Om vi tittar på att vi kanske har 300-400 program i vår programbank om man nu kan uttrycka det så jag menar utbildningen finns ju i stort sätt inte på nån av de här programmen, vi förutsätter att alla applikationer som finns där är självlärande och inte kostar. Jag menar om vi skulle säga att vi har 1000 användare av applikationen och vi har 2 timmars upplärnings tid är det 2000 timmar och 2000 timmar gånger timpenningen är typ 2miljoner eller vad det kan vara. Det är en stor del att det kan vara absolut enkelt att använda, det är otroligt viktigt så själva poängen där är att användningen är viktig eftersom att det är en kostnad.

Tror ni att tidigare kundskap av liknande program skulle vara till hjälp med inlärningen?

Ja det tror jag, jag har ju svårt att se att det här skulle vara, för nu pratar vi om i breda termer, det kommer inte vara så att alla kommer att använda det här det är jag rätt säker på att det kommer vara ett visst antal och samma sak där, nej jag tror inte att det skulle vara nått där för programmet i sig är bara ett verktyg och det är då design patterns och dem delarna som man skulle haft kundskap från innan och dratt nytta av.

Hur var det att använda programmet?

Jag tycker att resultatet saknar viss funktionalitet, det saknas också en motivering till resultatet.

Var svaret som programmet gav dig något du förväntat dig?

Ja det fanns nya men de patterns som fanns med är redan oavsiktligen använda redan asså de finns redan vi använder olika patterns fast vi har vissa andra omvägar runt om och liknande eller jag skulle till och med kunna säga så att vissa av dem här patterns som finns har en viss typ utav samtida användning, man kan implementera det ena mot det andra för de utesluter inte varandra i alla lägen. Ja det är nya men jag saknar fortfarande en referens till vad som gör ett pattern, pattern.

Kände du att din tidigare kundskap av liknande program hjälpte dig med just detta program?

Användargränssnittet var enkelt och förståeligt.

Tror du att ni skulle kunna ha användning för programmet?

Nej inte så länge fokus ligger på att försöka analysera fram korrekt pattern. Däremot så saknas det i så fall pattern uppslag mycket nästan gå åt andra hållet: det är dem här pattern som finns de används i de här scenarierna, alltså att det är mycket mer kopplat åt båda hållen på det sättet, då hade man sett nånting användbart men här så är det mer att det är inriktat på statistisk analysering av data för att få fram nånting och då är det så pass flexibelt i den riktiga världen att man kan inte riktigt lita på resultatet så därför tror inte jag att jag skulle kunna rekommendera det för användningen för andra arkitekter eller utvecklare.

Skulle det i så fall vara en förbättring om det var länkat i resultatet till vad de olika patterns gör?

Ja det skulle det vara.

Är det nått du känner att du vill tillägga eller om du har några frågor?

Nej ja tycker att mycket utav det här, det vi pratat om, har speglat den känslan jag har fått utav det. Jag tror att idag som sagt ligger det lite för mycket intresse på den matematiska delen för statistiskt plocka fram de här delarna istället för det som är runt omkring och då är det som faktiskt är användbart för användaren. Däremot så är det väldigt svårt att säga nånting om kärndelen i så fall som skulle vart den här insamlingen utav data för att plocka fram eftersom vi inte får nått bevis för att den tar fram rätt. Så att det är väl det.

Appendix B - Intervju 2

Intervju Företag 2, lead programmerare

Hur arbetar ni?

Vi jobbar lite olika på de olika delarna kan man säga, vi är några stycken som pair-programmerar ganska mycket det blir väl lite mer åt extreme programming hållet. Och sen så är det väl .. det blir väl så automatiskt vi har väl inte strukturerat det på nått speciellt sätt utan det blir väl mer att man roterar lite ändå. Hjälper varandra och kollar på varandras kod och sådär, men det är väl inte sådär superstrukturerat direkt.

Tänker ni mycket på utbyggbarhet?

Ja det gör man väll absolut. Det är väl en av dem sakerna man gör fel på kanske när man programmerar att man kodar lite för generellt, att man ska kunna bygga ut på det även om man kanske inte behöver det. Men det blir väll så det är roligare att koda på det sättet, man bygger nått generellt man kanske har nytta av nån gång i framtiden.

Har ni hört talas om design pattern recommender program förut?

Nej, det tror jag inte.

#förklarar vad ett design pattern recommender program gör

Skulle ni kunna tänka er att använda befintliga DPR program eller utveckla egna?

Ja vi skulle nog kunna kolla på det, det är svårt att säga när jag inte vet hur det är.

Tror ni att tidigare kundskap av liknande program skulle vara till hjälp med inlärningen?

Mm det tror jag absolut.

Tror ni att ett design pattern recommender program skulle fungera hos er?

Jao kanske, vi har ju inte använt det jätte mycket innan om man säger design patterns överhuvudtaget men absolut det skulle vara intressant att testa litegrann. Det är svårt att säga om det skulle funka.

Hur tror ni att inlärningen av programmet skulle fungera hos er?

Det skulle vara lätt, det var inte så klurigt, bara kryssa i lite nyckelord och sen var det klart i princip.

Hur var det att använda programmet?

Jo det var väl lite många kanske det var svårt att gå igenom alla när man skulle hitta nått man behövde just då kanske.

Var svaret som programmet gav dig något du förväntat dig?

Har inte så jättebra koll på design patterns i allmänhet och vet väll inte riktigt vad de betyder, det är väll nått man får läsa på och googla om dem när man sett vilka de var som de rekommenderade får man la kolla upp dem och hur man implementerade dem

Kände du att din tidigare kundskap av liknande program hjälpte dig med just detta program?

Jao, så märkvärdigt var det inte kryssrutor och en knapp.

Tror du att ni skulle kunna ha användning för programmet?

Ja de tror jag absolut man kan få, det gäller väll att börja använda lite design patterns också då eller kolla upp det när man väl har några problem som man ska lösa. Det är svårt att säga som sagt.

Finns det några förbättringar som du kan se??

Tror ju lite sån här standard grejer som man kanske behöver ofta eller nått kan man ju ha lite exempel och sånt tror jag kunde ha hjälpt lite. Det är väll alltid trevligt om man har sådana generella problem som man löser ofta eller nånting som man använder dom design pattterns till, mycket då exempel grejer hade ju varit bra tror jag. Men annars så vet jag inte riktigt. Svårt att säga såhär när man bara har testat det lite snabbt.

Appendix C - Intervju 3

Företag 3, Programmerare

#förklaring av vad arbetet går ut på

Hur arbetar ni?

Vi har ju en projekt owner/projekt ägare som äger projektet då som tar fram kraven som spelen ska uppfylla. Och sen har vi ett litet möte, som typ uppstarts möte på projektet där vi utformar kraven och kommer på hur vi ska göra om dem till spelmekanik typ. Och sen jobbar vi på i fem veckor. Vi jobbar i sprintar med två veckor i taget, vi kör SCRUM.

Tänker ni mycket på utbyggbarhet?

Ja lite, vi bygger ju flera spel utifrån en plattform kan man säga. Vi jobbar i unity så det är ganska lätt så man har ett projekt och sen gör man nått och så kan man återanvända massa assets typ grafik och kod och sånt.

Har ni hört talas om design pattern recommender förut?

Nej

#förklarar vad Design pattern recommender programmet gör

Skulle ni kunna tänka er att använda befintliga DPR program eller utveckla egna?

Till en början använda befintliga men sedan kanske göra egna skulle vara kul.

Tror ni att ett design pattern recommender program skulle fungera hos er?

Det hade nog funkat till en viss grad. Om det hade varit vettigt, om man kan lita på det. Vi använder inte så mycket design patterns så det kanske är en bra start. Om man får lite tips.

Hur tror ni att inlärningen av programmet skulle fungera hos er?

Jag tror inte vi hade behövt lägga så mycket tid på det. Kanske på att tolka det, vi hade nog lagt en dag på det.

Tror ni att tidigare kundskap av liknande program skulle vara till hjälp med inlärningen?

Ja om man har lite allmän kundskap av att programmera så är det väl nyttigt att förstå begreppen kanske. Man kanske hittar nått program som har med samma sak att göra alltså liknande DPR program. Det är väll mer allmän kundskap som gäller.

Hur var det att använda programmet?

Det var tråkigt. Det hade säkert varit kul om man hade gjort det på ett roligare sätt. Man skulle kunna gjort en gameifikation av det.

Var svaret som programmet gav dig något du förväntat dig?

Jag kände igen några patterns som den föreslog men vi använder inte patterns så mycket.

Kände du att din tidigare kundskap av liknande program hjälpte dig med just detta program?

Ja.

Tror du att ni skulle kunna ha användning för programmet?

Det tror jag säkert.

Finns det några förbättringar som du kan se?

Göra det lite roligare, mer som ett spel kanske.