GoF designmönster på webben med C#

Jasmin Bejtovic, jb223cp

Abstrakt

I programmering använder man ofta designmönster-koncept. Även om själva konceptet är känt sedan 80-talet, men den riktiga användningen av konceptet i mjukvaruutvecklingen ökade betydligt efter publicering av boken *Design Patterns: Elements of Reusable Object-Oriented Software* år 1994, av en så kallad "Gang of Four" (GoF). Boken anses vara absolut standard i branschen.

Utifrån designmönster beskrivna i boken försöker denna uppsats besvara frågan om praktisk betydelse av konceptet i dagens utveckling. Med tanken att konceptet undersöktes under en lång period av flera författare och teoretiker försöker uppsatsen undersöka det de oftast nämna hypoteserna i litteraturen.
Metoden för undersökningen är en litteraturstudie, och det analyserades en del valda böcker och artiklar. Sedan gjordes ett urval av nämnda hypoteser i båda kategorierna och utifrån hypoteserna undersöktes litteraturen ytterligare.

Resultatet av undersökningen visar att få tal hypoteser kan anses som tydliga styrkor respektive tydliga svagheter i utveckling med designmönster. Resultaten delades i tre kategorier: hypoteser vilka tillhör styrkorna av användning av designmönster-koncept, hypoteser vilka tillhör svagheter av designmönster koncept och hypoteser vilka är omöjliga att kategorisera.
Hypotesen att användning av designmönstrens ordförråd i utvecklingsprocessen underlättar kommunikation mellan flera utvecklare och avgränsar kommunikation på hög nivå utan att gå in på detaljnivå visar tydliga styrkor med användning av konceptet, medan hypotesen att användning av designmönster kan göra enkla problem mer komplexa visar sig sann och tillhör kategorin svagheter.

Nyckelord:

Designmönster, GoF, Gang of Four, C#, webbutveckling

1.Inledning

Designmönster i mjukvaruutveckling introducerades för första gången året 1987, när K. Beck och W. Canningham experimenterade med idéen om att applicera teorin om designmönster (först introducerad som arkitektoniskt koncept) i datavetenskap. [1]
Begreppet innebär att man katalogiserar olika typiska problem och deras typiska lösningar. [2]

Teorin om designmönster blev populär i datavetenskap efter publicering av boken *Design Patterns: Elements of Reusable Object-Oriented Software* året 1994, av så kallad "Gang of Four" (GoF). [3]
Boken anses som standard i konceptet om designmönster och grupperar tjugotre designmönster i tre kategorier: skapandemönster, beteendemönster och strukturmönster. Denna uppsats begränsas bara till designmönster angivna i denna bok.

Koncept av designmönster i utveckling har sina kritiker. Ett exempel är en artikel av Jeff Atwood, *Rethinking Design Patterns* [4] i vilken han påpekar två huvudsakliga problem med användning av designmönster.

1. I sin artikel hävdar J. Atwood att designmönster är en form av komplexitet. Atwood skulle hellre se utvecklare lösa mindre problem och utveckla mindre lösningar innan de fokuserar på den större bild i form av implementation av något komplext designmönster.
2. J.Atwood hävdar att om man har behov av att skriva designmönster kod för att lösa återkommande designproblem, tyder det på att programmeringsspråket i princip är brutet.

S. Rowe, webbdesigner och utvecklare, besvarar Atwoods artikel och hävdar i sin artikel *Are Design Patterns A Bad Idea?* [5] att huvudproblemet med Atwoods påstående är inte själva konceptet av designmönster i sig eller GoF boken, utan dålig implementation av specifika mönster.

Nämna exempel visar diametralt motsatta åsikter om delar av konceptet, vilket motiverar mig att undersöka ämnet och presentera resultatet av min undersökning.
Huvudsaklig fokus i denna uppsats kommer att ligga på diskussionen kring betydelsen av GoF designmönster. Designmönster kan variera i sin implementation och betydelse i olika programmeringsspråk, och därför kommer jag att begränsa min undersökning på .NET ramverk med C# som programmeringsspråk. Valet av ramverk och programmeringsspråk är baserat på personlig preferens, men resultatet av undersökning borde kunna appliceras till en viss del i flera ramverk och språk.

2. Bakgrund

2.1. Syfte

Syftet med denna uppsats är att undersöka praktisk betydelse av designmönster konceptet angivet i boken *Design Patterns: Elements of Reusable Object-Oriented Software* i utveckling av applikationer med .NET ramverk och C# som programmeringsspråk, och ge en tydligare bild över situationer när det är önskvärt att använda designmönster i problemlösning och när det är bra att undvika konceptet. Det vill säga, lägga tonvikten på styrkor och svagheter av designmönster konceptet.

2.2. Frågeställning

Med denna uppsats vill jag besvara följande fråga:

Vilka styrkor respektive svagheter finns det vid användning av designmönster beskrivna i boken Design Patterns: Elements of Reusable Object-Oriented Software i utveckling av webbapplikationer i .NET ramverk med C# som programmeringspråk?

2.3. Tidigare forskning

Designmönster-konceptet behandlas av ett större antal böcker och artiklar. Majoriteten av böckerna behandlar designmönster genom att undersöka egenskaper av specifika designmönster och presentera praktiska exempel med användning av behandlade mönster. I denna uppsats fokus ligger på fördelar och nackdelar med designmönster, och specifika egenskaper av enskilda designmönster är inte relevanta. Det finns en del artiklar, böcker och uppsatser som delvis behandlar uppsatsens område.

Ett bra exempel är i inledning nämnd artikel av K.Beck. [1]
M. Ali och M. Elish gjorde en undersökning om påverkande av designmönster på mjukvaruutveckling. De presenterade sina resultat i en uppsats [6] och deras arbete fokuserades mest på exempel i Java. Många klassiska mjukvaruutvecklings böcker behandlar designmönster på ett övergripande sätt i några delar. Några meriterande exempel är C.Larman bok om UML och designmönster [7], I Sommerville bok om mjukvaruutveckling [8] och R.C. Martin bok om agil mjukvaruutveckling. [9]

3. Metod

I följande kapitel förklaras metoderna som kommer att användas för att besvara frågeställningen. Inledande kapitel beskriver motiveringen bakom valda metoder. Kapitlet avslutas med en diskussion om metodens reliabilitet, extern validitet och intern validitet.

3.1. Val av metod

Litteraturstudie har valts som huvudmetod för denna uppsats. Detta innebär att en undersökning och analys av angivna exempel i litteraturen, och en jämförelse mellan olika hypoteser genomförs. Urvalet av litteratur och artiklar är gjort med sökning på nyckelord baserade på relevans till uppsatsens frågeställning.

Sökningarna är genomförda via den nätbaserade söktjänsten Google och några exempel på sökorden är: *advantages of design patterns, disadvantages of design patterns, design patterns in C#, Gang of Four, Impact of design patterns, Singleton, Decorator pattern* och så vidare.

3.1.1. Litteraturstudie

Syftet med litteraturstudien är att i första hand sammanställa hypoteser som är gemensamma för de flesta författare som förespråkar användning av designmönster i utveckling på webben i C#. Sammanställda hypoteser kan anses som styrkor eller svagheter i användning av designmönster. I första del av uppsatsen kommer bara styrkor av designmönster koncept behandlas, som nämnts frekvent i litteraturen samt i publicerade artiklar på webben.

I andra del av litteraturstudien samlas hypoteser från källor som kritiserar designmönster-koncept. På liknande sätt som sammanställningen av styrkor, genomförs sammanställning av svagheter av användning av designmönster. Avgörande faktor för val av hypoteser är liknande som faktorn för val av styrkorna.

3.2. Metoddiskussion

3.2.1. Extern validitet

Eftersom litteraturstudien har valts som huvudsaklig undersökningsmetod, blir extern validitet mindre kontrollerad om förutsättningarna ändras. Hypoteser i litteraturstudie är svåra att mäta men ett stort antal tillgängliga exempel (i princip, alla tillgängliga webbapplikationer med källkoden i C#) ger möjlighet att dra gemensamma slutsatser, vilka förhoppningsvis kommer att besvara frågeställningen.

3.2.2. Intern validitet

En kvalitativ undersökning av tillgängliga webbapplikationer kommer att genomföras, genom ett urval av de mest representativa exempel (noggrann källkritisk analys).

3.2.3. Reliabilitet

Analysen av representativa exempel och valda hypotesen från litteraturen är till viss del baserad på subjektiv tolkning. Själva frågeställningen ger möjlighet för mer flexibla och oscillerande resultat av eventuella upprepande undersökningar. Ambitionen är att uppmuntra till ytterligare forskning av dragna slutsatser. Tidigare forskning är bred och det gäller att lägga mycket tid på att avgöra vilka källor som är mest trovärdiga och att välja de mest representativa exempel som visar eller motbevisar hypoteser.

4. Sammanställning av resultat och analys

Resultatet som presenteras i arbetet är uppdelat i två delar. Första delen presenterar hypoteser som används frekvent i undersökta publikationer och artiklar. Fokus ligger på diskussionen kring användning av designmönster utifrån olika teoretikers perspektiv som granskar konceptet. I andra delen presenteras resultatet av analys med hjälp av praktiska exempel och litteraturstudier som förklarar och tydliggör angivna hypoteser.

4.1. Hypoteser

I avsnittet nedan följer en egen tolkning utifrån läsning av olika styrkor och svagheter som har identifierats som mest nämnda och omdiskuterade i litteraturen som legat till grund för arbetet.

4.1.1 Hypoteser som hänvisar till styrkor med användning av GoF designmönster

- Användning av designmönstrens ordförråd i utvecklingsprocessen underlättar kommunikation mellan flera utvecklare och avgränsar kommunikation på hög nivå utan att gå in på detaljnivå
- Användning av designmönster förbättrar kodens läsbarhet
- Användning av designmönster garanterar ökning i kodkvalitet

4.1.2. Hypoteser som hänvisar till svagheter med användning av GoF designmönster

- Användning av designmönster kan göra enkla problem mer komplexa
- Användning av designmönster är tecken på brist hos ett programmeringsspråk

- Vissa designmönster har negativ påverkan på prestanda
- Användning av olämpliga designmönster är en tydlig nackdel av designmönster koncept

4.2. Analys

4.2.1. Styrkor

4.2.1.1 Användning av designmönstrens ordförråd i utvecklingsprocess underlättar kommunikation mellan flera utvecklare och avgränsar kommunikationen på hög nivå utan att gå in på detaljnivå

Detta innebär att kommunikation inte sker enbart verbalt, utan med kod och kommentarer. Det ovannämnda påståendet är ett av de mest frekvent angivna bland teoretiker och utvecklare som stödjer användning av designmönster. Det innebär att det inte borde finnas några objektiva orsaker till varför påståendet inte skulle kunna appliceras på applikationer utvecklade i C#. I några av de behandlade artiklar och publikationer[7][8][9][11][14] går det att hitta stöd som pekar på att användning av designmönster kan vara ett praktiskt verktyg för dokumentation.

I sin undersökning från 1996 beskriver Marshal P. Cline följande [10]:

> *"Design patterns provide a standard vocabulary among developers. They provide a list of things to look for during a design review and a list of things that must be taught in a course on OO design. They communicate information between designer, programmer, and maintenance programmer at a significantly higher level than individual classes or functions" (Marshal 1996)*

Publiceringen av Marshal's undersökning är fyra år äldre än C# programmeringsspråk, men hans observationer är inte knutna till ett specifikt språk eller med språkets specifika funktionalitet, utan själva utvecklingsprocessen. Observationerna kan appliceras på C# även om ordförrådet kan variera mellan olika språk. Marshal ger svaret på detta i sin undersökning [10] i följande form:

> *"Even though the patterns and their names were different from those in standard use today, they were nonetheless uniform across the project teams, and this uniformity allowed the developers to communicate at a higher level of abstraction. (Marshal 1996)"*

Med tanke på att det inte finns betydande och omfattande källor som motbevisar eller vederlägger Marshal´s avhandling, är det rimligt att dra slutsatsen om att hypotesen i fråga faktiskt är en av de största styrkorna med användning av designmönster i utvecklingsprocess i C#.

4.2.1.2 Användning av designmönster förbättrar kodens läsbarhet

Med tanke på att analysen ovan visar att användning av designmönstrens ordförråd i utvecklingsprocessen är av de största styrkorna, kan det därmed konstateras att kodens läsbarhet ökar bland utvecklare som använder designmönster koncept. I sin artikel hävdar skribenten Harri Daniel följande [11]:

> " *Design patterns help to speed up software development by providing tested development paradigms. Software design involves considering small issues that might become visible later during the implementation process. Reusable design patterns help to correct subtle problems and enhance code readability."(Harri Daniel, 2011)*

Detta citat presenterar oproblematiskt kodens läsbarhet som en självklarhet, vilket är svårt att bevisa i praktik och är mer en bedömningsfråga än ett empiriskt bevisat resultat. Förbättrad läsbarhet kan tolkas som en bieffekt av användningen av designmönster konceptet, men endast om designmönster konceptet implementerar ett visst specifikt ordförråd. Utvecklare som är vana vid konceptet kan lättare läsa koden som följer riktlinjer av specifika designmönster, dock blir det svårare för utvecklare som inte är vana vid konceptet. Därmed kan man inte som Harri Daniels hävdar påstå att användning av designmönster ökar läsbarheten per automatik. Namngivning av variabler och instanser vilket är identiskt som vissa designmönster (observer, facade, strategy och så vidare) kan definitivt i vissa fall öka kodens läsbarhet. Huvudfrågan kvarstår dock. Ökar användning av designmönster kodens läsbarhet på grund av förbättrad kodstruktur eller är kodens förbättrade läsbarhet en bieffekt av användning av specifikt ordförråd? Med tanke på att frågan inte är besvarad kan inte hypotesen räknas som en styrka i användningen av designmönster.

4.2.1.3 Användning av designmönster garanterar ökning i kodkvalitet

Ovannämnda hypotes lyfts fram i nästan alla undersökningar som används i denna uppsats om designmönster koncept. Den mest omfattande undersökningen om designmönstrens påverkan på kodkvalitet presenteras av forskarna A.Mawal och M Elishi deras publikation [6] som begränsas till enbart fyra aspekter av kodkvalitet och bara en grupp av designmönster. Deras arbete ger dock inga definitiva slutsatser och uppmuntrar till vidare undersökning.

Mawals och Mahmouds arbete kan bäst appliceras på några av aspekterna på kodkvalitet. En av dessa aspekter som de nämner är prestanda, vilket kommer att behandlas i ett senare avsnitt av denna uppsats. Professor S. Ramasamy, G. Jekese och C. Hwata från CRM University diskuterar i sin publikation kring inflytande av designmönster på mjukvaruutveckling och menar att [12]:

> *Many studies in the literature are based on the premise that design patterns improve the quality of object-oriented programs. Yet, some studies suggest that the use of design patterns does not always result in "goo" designs. For example, a tangled implementation of patterns impacts negatively the quality that these patterns claim to improve. Also, patterns generally ease future enhancement at the expense of simplicity. Thus, evidence of quality improvements through the use of design patterns consists primarily of intuitive statements and examples. There is little empirical evidence to support the claims of improved reusability, expandability and understandability as put forward in when applying design patterns. Also, the impact of design patterns on other quality attributes is unclear. (Ramusamy, Jekese, Hwata 2015)*

Beskrivningen ovan visar att hypotesen i stort sett är abstrakt, vilket betyder att det finns behov av att tydliggöra definitionen av kodkvalitet. Kodkvalitet är ett svårdefinierat begrepp och beskrivningen som bäst motsvarar behoven i denna uppsats är definierad i artikel av D. Spinellis [13]. Spinellis föreslår femton regler som ska hjälpa till och definiera kodkvalitet. En av dessa regler är regel nummer två. Den uppmuntrar användning av beskrivande namngivning. Eftersom designmönster konceptet uppmuntrar till användning av eget ordförråd, vilket är beskrivande i sig själv, kan den regeln bekräfta att användning av designmönster garanterar ökning i kodkvalitet.

Regel nummer tre behandlar kommentarer och dokumentation, vilket är ett plus till för designmönster koncept. Utvecklare med kännedom om designmönster kan läsa kod skriven med hjälp av någon av designmönstren som att koden är självdokumenterad.
Användning av designmönster garanterar att reglerna nummer sex och nummer nio blir uppfyllda. Speciellt viktigt är regel nummer 9 :

> *"Do similar things in similar ways. If you're developing a routine whose functionality resembles that of an existing routine, use a similar name, the same parameter order, and a comparable structure for the code body. The same rule applies to classes: Give the new class similar fields and methods, make it adhere to the same interface, and match any new names with those already used in similar classes. Make the order and number of case statements or if elseblocks follow the corresponding definition in the specifications you're using. Keep unrelated items in alphabetical or numerical order." (Spinellis 2014)*

Användning av designmönster försäkrar per automatik att föregående regel är uppfylld eftersom designmönster konceptet är baserat på återanvändning av redan definierade lösningar på specifika problem.

Analysen förklarar att det inte finns några definitiva svar om att användning av designmönster garanterar ökning i kodkvalitet. Några aspekter av kodkvalitet är påverkade positivt och några negativt.

4.2.2 Svagheter

4.2.2.1 Användning av designmönster kan göra enkla problem mer komplexa

Denna hypotes kan också hittas i litteraturen under begrepp av överanvändning. [12] Utvecklare med kännedom om designmönster konceptet har tendens att överanvända konceptet och det kan leda till mer komplexitet. [14] [10] Nedan följer två beskrivningar om överanvändning och dess konsekvenser.

> *One of the things that can be a problem is that people can think that patterns are unreservedly good and that a test of a program's quality is how many patterns are in it. This leads to the apocryphal story of a hello world application with three decorators, two strategies, and a singleton in a pear tree. Patterns are not good or bad—rather, they're either appropriate or not for some situations. I don't think it's wrong to experiment with using a pattern when you're unsure, but you should be prepared to rip it out if it doesn't contribute enough. (Fowler 2003)*

> *Design patterns are overly hyped. They cannot possibly deliver the benefits that some have naively stated. Almost everyone in the design patterns community has been concerned about this for some time, but unfortunately, some consultants either do not understand the limits of the technology or worse. (Marshal 1996).*

Teoretiker som försvarar konceptet av designmönster påpekar dock att ovanstående hypoteser i stor grad påverkas av utvecklares erfarenhet. De menar därmed inte att användning av designmöster inte samtidigt kan öka komplexitet. Analysen av deras undersökning, och den som framgår i denna uppsats visar att föregående hypotes är sann och ingår i gruppen som räknas till svagheter av användning av designmönster.

4.2.2.2 Användning av designmönster är tecken på brist hos ett programmeringsspråk

Detta påstående var ett av de första som riktade kritiken mot designmönster konceptet. Det är nämnt i J.Atkwood's artikel [4] för första gången. Flera andra teoretiker nämner Atkwood's påstående. I sin artikel [5] analyserar S.Rowe några delar av Atkwoods artikel och sammanfattar det på följande vis:

> *"I think Jeff makes some good points but misses the mark. First, his comment about it being a language issue is interesting, but more often than not we don't have a choice of languages. Not just because our bosses dictate the language, but because most languages have serious weaknesses and even if they solve our pattern problem, they may not be robust enough, have the right libraries, be what the team knows, etc. "*

Hur appliceras ovanstående påstående på ett programmeringsspråk som C#? Analysen av specifika designmönster visar på att de används på flera olika sätt i C#. Singleton designmönster, som begränsar antalet instanser av en klass till ett objekt, [15] har ett alternativ i användning av

statiska klasser. Statiska klasser är lättare att implementera och mängden av kod blir mindre, men där slutar alla fördelar med statiska klasser över singleton designmönster. Singleton designmönster kan ärva från andra klasser, kan implementera interface, andra klasser kan ärva från singleton och singleton klass kan ha konstruktor, vilket inte en statisk klass kan. Allt tyder på att implementeringen av singleton designmönstret är fördelaktigt över användning av statisk klass i applikationer när det bara behövs ett objekt instans av en klass. Från Atwood's ståndpunkt som han för fram i sin artikel kan denna brist ses som brist i programmeringsspråk.

Ett annat exempel är användning av arkitekturella mönster MVVM (model-view-viewmodel) i webbapplikationer i C#. ViewModel klasser implementerar ofta model klasser med syftet om att försäkra sträckbarheten av vanliga attributer i model klasser. Detta konceptet påminner om dekoratör designmönster [16]. Skillnader är att dekoratör designmönster implementerar abstrakt klass, medan model klasserna i MVVM mönster ofta är konkreta. Composite designmönster [17] används för att implementera nästlade vyer i MVC och MVVM arkitektur.

Nämnda exempel visar fördelar och nackdelar av C# när det gäller användning av designmönster. Trenden visar att konceptet av designmönster hjälper programmeringsspråk att adoptera nya funktionaliteter vilket i stort sett bekräftar Atwood's hypotes. Det visar också att trenden är till fördel för användningen av designmönster. Det vill säga, i C# finns det många exempel på användning av designmönster, och att man i framtiden kan förvänta sig fler. Allt tyder på att implementeringen av designmönster tyder på flexibilitet av ett specifikt programmeringsspråk, och att Atwood's påstående inte kan appliceras på modern utveckling i C#.

4.2.2.3 Vissa designmönster har negativ påverkan på prestanda

Hypotesen ovan behandlar en ny aspekt av designmönster. Även om läsbarhet, återanvändning, själva lösningen och andra faktorer som rör själva programmeringskoden, är det intressant att undersöka om vissa designmönster har påverkan på prestanda. A.Mawal och M. Elish som vidare hänvisar till en undersökning av Rudzki från 2005 nämner tre olika designmönster och dess påverkan på prestanda [6]. Deras tolkning av Rudzkis undersökning är följande:

> *Rudzki studied the difference between the impacts of two different design patterns on the performance of a software system. He chose the Command and the Façade design patterns in his study. The reason for choosing these two patterns is that they have similar functionality so they can be used alternatively. By running the system in 9 different test cases, he found that the Façade pattern performed better that the Command pattern. (Mawal och Elish 2011)*

Rudzkis undersökning är gjort på exempel med fler-lager J2EE applikation. I litteraturen saknas undersökningar gjorda på applikationer skrivna i C#.

Rudzkis undersökning visar bara att ett designmönster presterar bättre än ett annat designmönster. Det saknas användningsfall om när designmönster implementeringen visar negativ påverkan på prestanda i jämförelse med applikationer som inte implementerar några designmönster. Detta innebär att det inte är rimligt att dra definitiva slutsatser om huruvida användning av designmönster påverkar prestanda negativt.

4.2.2.4 Användning av olämpliga designmönster är tydligt nackdel på designmönster koncept

Användning av designmönster handlar om att lösa specifika problem. Om utvecklare har kännedom om vissa designmönster men saknar kunskap om flera andra designmönster, kan det leda till en situation där utvecklare försöker implementera designmönster som inte är lämplig för specifik problem. I sin publikation om designmönstrens inflytande på mjukvaruutveckling hävdar Professor R.Subburaj, G. Jekese, C. Hwata följande [12]:

> *[...]There is a tendency for some developers to adopt a tunnel vision where they would try to apply patterns that were inappropriate, simply because they were familiar with the patterns. (Subbaraj, Jekese och Hwata 2015)*

och fortsätter med citat från Fowels artikel från 2003 [14]:

> *[...]Reactor pattern may be an inefficient event demultiplexing model for a multi-processor platform since it serializes application concurrency at a fairly coarse-grained level. (Fowel 2003)*

Föregående undersökning pekar på ett tydligt problem, men problemet uppstår mer som brist på kompetensen hos utvecklare som implementerar designmönster än som nackdel eller svaghet i konceptet. Med tanke på detta, kan i avsnittet rubricerade hypotes anses som oklar och representerar inte en svaghet av konceptet. I denna uppsats är hypotesen kategoriserad bland hypoteser som varken är styrkor eller svagheter av designmönster konceptet.

4.3. Kategorisering av hypoteser

Hypoteser		
Styrkor	Kan inte appliceras	Svagheter
• Användning av designmönstrens ordförråd i utvecklingsprocessen underlättar kommunikation mellan flera utvecklare och avgränsar kommunikation på hög nivå utan att gå in på detaljnivå	• Användning av designmönster förbättrar kodens läsbarhet • Användning av designmönster garanterar ökning i kodkvalitet • Användning av designmönster är tecken på brist hos ett programmeringsspråk • Vissa designmönster har negativ påverkan på prestanda • Användning av olämpliga designmönster är tydligt nackdel på designmönster koncept	• Användning av designmönster kan göra enkla problem mer komplexa

5. Diskussion och slutsatser

Efter att ha undersökt litteratur om dessa designmönster, kunde det konstateras att det finns väldigt få definitiva slutsatser som tydligt visar på fördelar eller nackdelar av användningen av designmönster med utveckling i C#.
Undersökningen visade att nästan alla hypoteser, frekvent nämnda i den sammanställda litteraturen, har tillräckligt många aspekter som motbevisar hypotesen och dessa aspekter kan inte ses som ett undantag.
Det finns en hypotes som tydligt tillhör styrkor av designmönster-konceptet och en hypotes som tydligt tillhör svagheten av konceptet.

5.1 Hypoteser

HYPOTES 1: Användning av designmönstrens ordförråd i utvecklingsprocessen underlättar kommunikation mellan flera utvecklare och avgränsar kommunikation på hög nivå utan att gå in på detaljnivå

HYPOTES 2: Användning av designmönster kan göra enkla problem mer komplexa.

6. Slutligen

Jag vill tacka min handledare Mats Loock och Johan Leitet vid Linnéuniversitetet och klasskamraterna Roy Nilsson, Mikael Eriksson, Mattias Wikström, Robin Karlsson och Matilda Ingman för förslag på förbättringar av uppsatsen.

7. Referenser

[1] K. Beck, W. Cunningham (1987). *Using Pattern Languages for Object-Oriented Programs.* [www]. Hämtad 2016-04-21 från http://c2.com/doc/oopsla87.html

[2] Wikipedia. [www]. Hämtad 2016-04-01 från https://sv.wikipedia.org/wiki/Designmönster

[3] E. Gamma, R. Helm, R. Johnson och J. Vlissides . *Design Patterns Elemenths of Reusable Object-Oriented Software, Addison-Wesley 1994, New York.*

[4] J. Atwood (2007). "*Rethinking Design Patterns.", Coding Horror* [www]. Hämtad 2016-03-31 från http://blog.codinghorror.com/rethinking-design-patterns/

[5] S. Rowe (2007). "*Are Design Patterns A Bad Idea?," MSDN Blogs* [www]. Hämtad 2016-04-01 från https://blogs.msdn.microsoft.com/steverowe/2007/07/11/are-design-patterns-a-bad-idea/

[6] Mawal Ali and Mahmoud O. Elish (2011). "*A Comparative Literature Survey of Design Patterns Impact on Software Quality ", Information & Computer Science Department King Fahd University of Petroleum & Minerals Dhahran, Saudi Arabia* [www]. Hämtad 2016-05-19 från *https://www.computer.org/csdl/proceedings/icisa/2013/0602/00/06579460.pdf*

[7] C. Larman . *Applying UML and Patterns, Prentice Hall 2013, Westford.*
[8] I. Sommerville. *Software Engineering, Addison-Wesley 2001, Harllow.*
[9] R. C. Martin . *Agile Software Developement, Prentice Hall 2003, New Jersey.*

[10] Marshal P. Cline (1996). *"The Pros and Cons of Adopting and Applying Design Patterns in the Real World.", Research Gate* [www]. Hämtad 2016-05-14 från

https://www.researchgate.net/publication/220424606_The_Pros_and_Cons_of_Adopting_and_Applying_Design_Patterns_in_the_Real_World

[11] D. Harri (2011). "*Benefits Of Design Patterns. ", BenefitsOf* [www]. Hämtad 2016-05-16 från *http://benefitof.net/benefits-of-design-patterns/*

[12] R.Subburaj Professor, Gladman Jekese, Chiedza Hwata (2015). "*Impact of Object Oriented Design Patterns on Software Development ", International Journal of Scientific & Engineering Research, Volume 6, Issue 2, February-2015* [www]. Hämtad 2016-05-19 från *https://www.researchgate.net/publication/273457811_Impact_of_Object_Oriented_Design_Patterns_on_Software_Development*

[13] D. Spinellis (2014). "*15 Rules for Writing Quality Code. ", Informit* [www]. Hämtad 2016-05-16 från *http://www.informit.com/articles/article.aspx?p=2223710*

[14] M. Fowler (2003). "*Design Patterns ", IEEE SOFTWARE* [www]. Hämtad 2016-05-17 från *http://martinfowler.com/ieeeSoftware/patterns.pdf*

[15] Wikipedia. [www]. Hämtad 2016-05-14 från https://sv.wikipedia.org/wiki/Singleton

[16] Wikipedia. [www]. Hämtad 2016-05-14 från https://en.wikipedia.org/wiki/Decorator_pattern

[17] Wikipedia. [www]. Hämtad 2016-05-14 från https://en.wikipedia.org/wiki/Composite_pattern

[16] P. Wendorff (2001). *"Assessment of Design Patterns during Software Reengineering: Lessons Learned from a Large Commercial Project" , IEEE*