En redogörelse för utvecklingsprocessen av ett bildredigeringsprogram

Av:
Anton Hagvall
Hanna Jacobsson
Niclas Johansson
Kristina Markan

28 maj 2017

LSP310 Kommunikation och ingenjörskompetens
TDA367 Objektorienterat Programmeringsprojekt
Civilingenjör Informationsteknik
Chalmers Tekniska Högskola

Sammandrag

Syftet med denna rapport är att redogöra för utvecklingsprocessen av bildredigeringsprogrammet Layers. Teoridelen beskriver några av de metoder som var inblandade i att skapa programmet och krävs för att resten av rapporten ska förstås. Den scruminspirerade arbetsmetod som användes beskrivs i metoddelen av rapporten.

I resultatet beskrivs den slutgiltiga designen av programmet. Här beskrivs även de olika nivåer av kontroller som används i kodstrukturen och hur de är kopplade med resten av programmet.

Slutligen diskuteras de val som gjorts i framställningen av gränssnittet samt kodstrukturen, hur de påverkat den slutgiltiga produkten samt hur de har anpassats för användaren. Här utvärderas även den arbetsmetod som använts och vikten av att skapa dokumentation kontinuerligt i processen.

Innehåll

1 Inledning

Hur många bilder tar du varje år? Önskar du att de ibland såg lite annorlunda ut? Sedan smartphonen introducerades tar de flesta människor inte bara bilder då och då, utan dagligen. Att få dessa bilder att se ut som man vill kan ta både färdighet och kunskap, vilket kan leda till en tidskrävande process. Layers är skapat för snabb och effektiv fotoredigering. Ingen tid borde slösas på att förstå krångliga program. Lägg till filter eller text, öka skärpan eller oskärpan. Alla grundläggande verktyg finns till hands precis där du behöver dem.

1.1 Bakgrund

Nu för tiden när en stor del av samhället har blivit digitalt, så har bildbehandling flyttat ut från mörkrummen och in i datorn, där långt många fler har möjlighet att använda sig av tekniken. Bilder sprids på ett helt annat sätt än förr, då de mest lyckade bilderna hängdes upp på väggen men de allra flesta låg kvar undangömda i ett fotoalbum. Idag är bilder en grundpelare i många sociala medier. Att behandla en bild innan den läggs upp på nätet kan göra så att den syns bättre genom bruset, det har visat sig att en behandlad bild har 21% större chans att bli visad än en som inte är det[1].

Med utvalda guldkorn från konventionella program är tanken att skapa något som kan användas av en vanlig användare, utan en hög inlärningskurva. Eftersom inlärningskurvan inte är hög innebär det också att taket inte kommer vara lika högt som de avancerade program som finns ute på marknaden. Tanken är att skapa ett program som stödjer den kreativa processen hos en vanlig användare, genom att hålla det enkelt men ändå ge god möjlighet till att utforska olika arbetsflöden[9].

Den bildbehandling som kommer att vara tillgänglig i programmet är främst baserad på linjär algebra och matriser[6]. Genom att multiplicera enskilda pixlar i bilden samt dess närliggande grannar med speciella matriser kan man få effekter såsom skärpa, oskärpa, intensifiering, samt att genomföra kantigenkänning. Utöver detta kommer vi även ge användaren möjlighet att förändra färgerna i bilden, vilket görs med justering av pixlars **Red-Green-Blue (RGB)**-värden. All funktionalitet kommer att presenteras för användaren i ett enkelt och stilrent grafiskt användargränssnitt med flera vägar att gå för att nå målet, bland annat **menyfält**, **toolbar** och **keybord shortcuts**.

1.2 Syfte

Syftet med rapporten är att beskriva utvecklingen av bildredigeringsprogrammet Layers samt analysera de metoder som använts i processen. Layers är en **Java-applikation** vars **gränssnitt** är utvecklat med **JavaFX** och som är skapad för **desktopformat**. Rapporten analyserar de designval som gjorts i gränssnittet och de val som gjorts angående funktionaliteten.

2 Teori

I utvecklingen av programmet har många olika tekniker använts. Tre av de viktigaste beskrivs nedan.

2.1 Domain-driven Design

Domändriven design handlar om att få ut så mycket som möjligt utav personer med olika expertkunskaper och hur de samverkar som bäst[2]. I det här sammanhanget handlar det om ett samarbete mellan experter inom domänen och experter inom design och mjukvaruutveckling. För att underlätta kommunikation krävs ett gemensamt språk som är förståeligt för alla inblandade parter. Medvetenhet om vilka sorters ord som används från både designers och utvecklare när de ska kommunicera med modellen är viktigt. Likaså är det även viktigt att personerna med domänexpertis tänker på vad de använder för ord när de kommunicerar.

Under utvecklingsprocessen finns det fyra huvudroller – domänexperter, designers, utvecklare och de slutgiltiga användarna. Domänexperter, designers och utvecklare ska använda sina kunskaper tillsammans för att skapa en så bra slutprodukt som möjligt för användarna. Allt börjar med att en specifik domän har ett problem som behöver design- och utvecklingsexpertis för att kunna lösas. För att lättare veta att alla förstår och är delaktiga i utvecklingen utgår man ifrån en domänmodell. Domänmodellen byggs upp av designern tillsammans med de andra medverkande. I domänmodellen ska inga programmeringstermer eller liknande förekomma, utan istället ska applikationens huvuddelar synas klart och tydligt oberoende om man är utvecklare eller domänexpert. Därför används ord som tillhör domänen[2].

Domändriven design är en iterativ process. Modellen är till för att kunna ändras i iterationer. Utgångsläget är att första modellen ska vara så välfungerande som möjligt. Om något inte fungerar vid första försöket kan det utvärderas och nya lösningar kan utvecklas[3].

2.2 Model-View-Controller

Model-View-Controller (MVC) är ett designmönster som är vanligt förekommande i program som innehåller någon typ av grafisk representation. MVC introducerades 1979 av Trygve Reenskaug på Xerox Palo Alto Research Center och separerar datan, den grafiska representationen och användarinteraktionen i ett program[4].

Det finns många olika varianter av MVC, som t.ex. **MVP**, **HMVC** och **MVVM**, men grundprincipen är densamma. Grundprincipen är att modellen, det grafiska användargränssnittet(vyn) och kontrollern ska kunna ändras separat från varandra utan att de andra delarna påverkas av ändringen. Fördelen blir att

flera gränssnitt eller flera olika sätt att samla användarinput kan användas till samma modell[4].

Modell-delen av MVC innehåller all data, tillstånd, och domänlogik. Modellen ska kunna presenteras och kontrolleras av användaren på olika sätt utan att valet påverkar modellens tillstånd. En modell får ofta sin information från en kontroll och uppdateras därefter. Det finns oftast endast en modell till flera vyer och kontroller[5].

Vy-delen fokuseras på allt visuellt som ska visas i programmet. Olika vyer används ofta beroende på vilken enhet programmet ska användas på. En vy får uppdateras utifrån modellen och beräkningarna som sker där.

Kontroll-delen är den del som samlar in information från användaren. Olika kontroller kan hantera olika sorters indata från användaren och kan ha påverkan på både modellen och vyer.

Tillsammans kommunicerar delarna på ett sätt som skapar ett stabilt men samtidigt flexibelt program.

2.3 Pixelmanipulation

En av de viktigaste teknikerna som används kallas Kernel Convulation. Kernel Convulation är en teknik som med hjälp av en viktad matris, även kallad en kernel, gör olika förändringar på en bild. En kernel kan se ut på olika sätt men ett av de vanligaste fallen är en 3x3-matris där alla värdena i matrisen är samma tal. Matrisen placeras sedan på en grupp av pixlar i en bild så att den pixel som man vill göra förändringen på ligger i mitten. Det algoritmen gör är att ta de pixlar som täcks av matrisen och summerar dem med den viktning som var angiven i matrisen. Pixlar som har högre viktning, vanligtvis de pixlar som ligger närmast den pixel som ska förändras, får ett högre värde när pixlarna summeras. Efter finns viktad summa för pixlarna som ligger i matrisen så tar algoritmen ett genomsnitt genom att dela summan med antal pixlar och dess viktningar. Denna beräkning resulterar i mittenpixelns slutgiltiga värde[6].

Kernel Convulation är inte bara användbart för estetiska förändringar. Ett av de nya användningsområdena för det är kantigenkänning, som främst används inom artificiell intelligens samt neurala nätverk. Kantigenkänning använder sig av en speciell matris som är viktad som en kant istället för en cirkel, för att leta upp de områden som har en hög kontrast mellan pixlar nära varandra. Detta kan till exempel vara en stol mot en vit vägg. Algoritmen använder kantdatan för att med hjälp av maskininlärning och andra tekniker försöka ta reda på vad det är för föremål på bilden. Tekniken kallas computer vision och är ett av de viktigaste och svåraste områden som finns inom datavetenskapen idag[7].

3 Metod

Den metod som har använts vid utvecklingen av programmet bygger till stor del på scrumprinciper som utvecklades av Ken Schwaber och Jeff Sutherland[8].

3.1 Scrum

Scrum är ett **ramverk** som beskriver ett vanligt sätt att utveckla storskalig programvara på ett effektivt sätt. Scrum går ut på att man delar upp en utvecklingsprocess i så kallade sprintar på två veckor. I början av varje sprint så skapar teamet en lista på de saker som ska göras och sätter en prioritering på listans punkter. Punkterna på listan delas mellan medlemmarna i scrumteamet, som själva tar på sig olika uppgifter att arbeta med. I slutet av varje sprint utvärderar teamet hur arbetet har gått och vad som kvarstår inför nästa sprint.

3.2 Genomförande

Programmets utvecklingsprocess använde en scrumliknande struktur. Utvecklingsprocessen var indelad i fyra sprintar med något varierande längd.

3.2.1 Sprint 1: Pappersdesign

Under sprint ett gjordes pappersutvecklingen av programmet. Fokus under sprinten låg på att designa både en domänmodell samt att skapa en grafisk profil för programmet. Många förslag gicks igenom på kort tid för att skapa en grund inför implementationen.

3.2.2 Sprint 2: Gränssnitt

Sprint två dedikerades åt att skapa gränssnitt och kärnfunktionerna i programmet. Under sprinten skapades majoriteten av de grafiska elementen samt de kärnfunktioner som behövdes för att kunna implementera transformationerna på ett lätt och effektivt sätt. Under sprinten färdigställdes alla de funktioner som berörde inladdningen av bild samt de klasser som hanterar lagren implementerade.

3.2.3 Sprint 3: Test och utvärdering

Sprint tre användes till att utvärdera den design och **systemarkitektur** som implementerats under sprint två. Här var målet att se om vår design skulle fungera i praktiken och leda till att programmet utnyttjade principerna bakom **easy extensibility**. Under sprint tre testades även användarvänligheten i gränssnitt för att se om det fanns möjligheter att effektivisera användandet.

3.2.4 Sprint 4: Slutimplementation

Sprint fyra användes för slutimplementationen. Under sprinten så implementerades alla de filter och transformationer som användaren har tillgång till genom gränssnitt. Alla klasserna byggde på den struktur vi skapade i sprint två och utnyttjade det vi lärde oss i sprint tre för att på ett så effektivt sätt som möjligt implementera många funktioner på kort tid.

4 Resultat

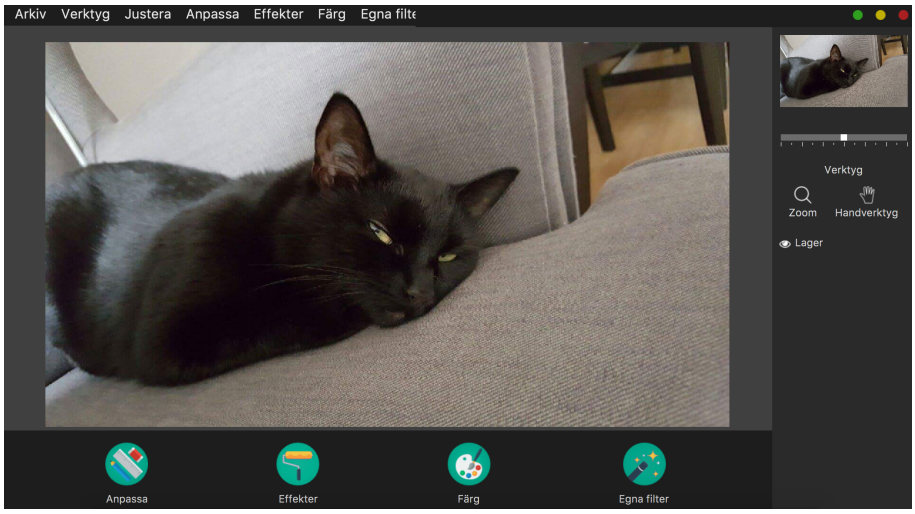
Här följer resultatet av den process som skapat Layers med fokus på gränssnitt och implementationen.

4.1 Gränssnitt

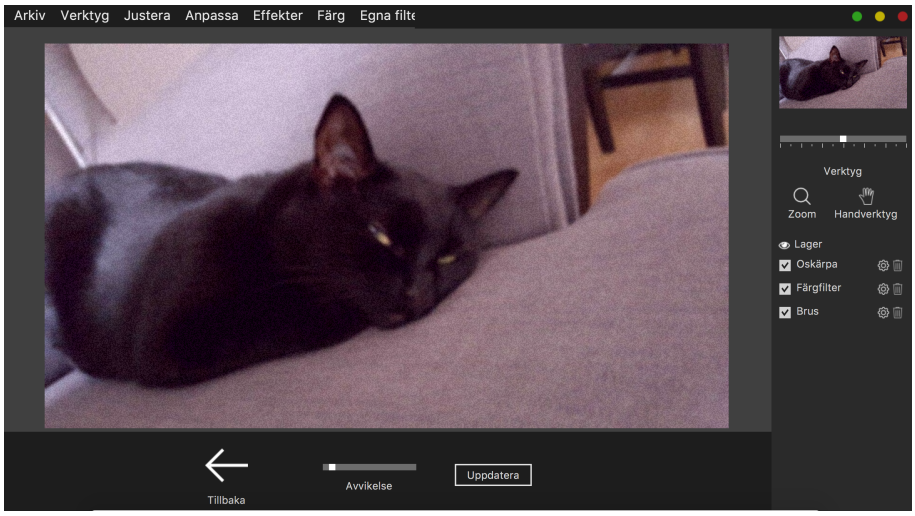
Layers har ett simpelt användargränssnitt som består av en enda vy. Detta leder till minimal navigation i programmet. Största delen av skärmen tas upp av bilden på vår **center stage**. Under bilden finns en toolbar där de vanligaste funktionerna kan nås. När en funktion är vald fylls toolbaren istället med vyn som visar inställningarna för funktionen.

Ett annat sätt att finna funktioner i programmet är genom den övre menyn. Menyn är uppdelad i flera olika kategorier och är en samling av alla funktioner som programmet erbjuder. Även när en funktion från menyn väljs så byts toolbaren ut mot funktionens inställningar.

Till höger finns en spalt för lagerhantering och zoomning. Alla lager visualiseras i en lista, och kan tas bort permanent eller göras osynliga för stunden. Zoomningen hanteras via en **slider**, och översikt skapas genom bildminiatyren som finns överst i denna spalt. Själva navigeringen i bilden sköts genom att användaren först väljer verktyget flytta och sedan drar bilden åt det håll användaren vill att bilden ska förflyttas. Både zoomning och förflyttning kan även göras genom att klicka på någon av de verktygsikoner som finns i spalten.



Figur 1 *Gränssnittet med en inladdad bild.*



Figur 2 *Gränssnittet med en inladdad bild med ett antal lager inladdade samt verktygsfältet med inställningar för lagret*

4.2 Implementation

Programmets designstruktur bygger till stor del på principerna om easy extensibility [3] . Denna princip är implementerad med grundtanken att det ska vara lätt att i framtiden utöka befintlig funktionalitet utan att tvingas skriva om allt för stora mängder kod.

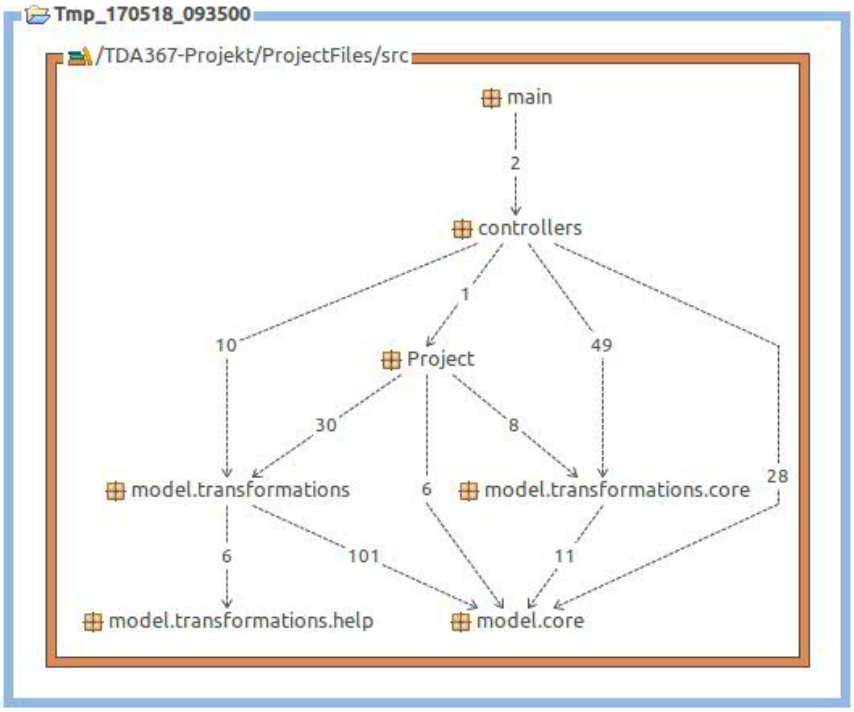
I programmet så genomförs detta genom att implementera en struktur där de olika bildförändringarna är separerade från grundfunktionerna som att öppna

bild eller spara projektet. För att skapa detta betetende är programmet uppdelat i tre nivåer. De tre olika nivåerna arbetar **linjärt beroende** av varandra.

Den första nivån är kontroller som har i uppgift att göra om den data som producerats i modellen och visa den i ett gränssnitt som är lätt för användaren att använda.

Den andra nivån i programmet är model.core. model.core har som huvuduppgift att hålla koll på de olika funktioner som är implementerade och koppla rätt funktion med rätt plats i kedjan av kommandon. I model.core finns även de klasser som ansvarar för att spara och ladda projekt samt de klasser som sparar data på hårddisken.

Tredje nivån är model.transformations. Det är den lägsta nivån som innehåller klasserna som gör de förändringar användaren kan applicera på bilden. Alla dessa förändringar har sina egna klasser som implementerar Layerable. Layerable är ett **interface** som används för att gruppera in alla de funktioner som kan läggas i lager och samla deras funktionalitet under ett lättanvänt **gränssnitt**.



Figur 3 *Structure Analysis (STAN)-analys som utfördes på programmet och hur strukturen undviker cirkelberoenden.*

5 Diskussion

Utifrån resultatet finns ett antal punkter som kan diskuteras ytterligare. Här redovisas en mer djupgående diskussion kring dem.

5.1 Metoder

Mot slutet av programmets utvecklingsprocess så märktes det att arbetetsfördelning mellan de fyra sprintarna inte var så effektivt som det borde varit. Som arbetet flöt på gjorde det att de fyra sprintarna hade liknande tidsomfång men väldigt olika arbetsbelastning. Exempel på det är att i sprint ett och tre producerades en väsentlig del mindre kod än i sprint två och fyra. Det skapade ett oflyt i programmets utveckling, skapade problem med kodkontinuitet och skapade även stor densitetsskillnad mellan de fyra olika sprintarna. Denna skillnad gjorde att vissa delar av dokumentationen hamna efter under de mindre intensiva sprintarna och att de inte uppnådde sin fulla potential under de mer belastade sprintarna. Under de mindre belastande sprintarna så tappade även utvecklingen **flow** och kravspecifikationerna ändrades väldigt lite vilket ledde till låg utvecklingshastighet.

Det var dock inte bara nackdelar då framförallt sprint tre och arbetet med att utvärdera systemarkitekturen skapade bra förutsättningar för effektivt arbete under sprint fyra vilket var något som inte vore möjligt annars.

5.2 Designval i modellen

Den slutgiltiga designmodellen levde upp till de krav som sattes vid arbetets start. Kraven byggde till stor del på principen om easy extensibility, en princip som värderades högt under programmets utvecklingsprocess. Dock så planerades det inte för i de tidigare faserna av programmet för möjligheten att skapa utbyggda **API**:er för att tillåta tredjepartsutvecklare att skapa sin egen funktionalitet på den grund som programmet utgör. Detta är något som borde tänkts på tidigt för att på en ännu större skala sträva mot principen av easy extensibility. Ett sätt att genomföra detta hade varit att genom en mer dynamisk process tillåta användare att registrera sina klasser till huvudprogrammet och att sedan generera de grafiska element som är bundna till de olika filtren. Detta hade kunnat uppnås med en **handler**-klass som hanterar de filter som finns i programmet och genom att tillåta filtren att själva definiera vilken kategori de tillhör.

5.3 Designval i gränssnittet

Redan tidigt i designprocessen bestämdes att fokus skulle ligga på att skapa ett enkelt och lättförståeligt gränssnitt. Prioriteringar om vad som skulle synas och vad som skulle läggas i underkategorier behövdes därför göras. I slutändan gjordes valet att dölja de flesta funktioner som inte hade med lagerhantering eller

navigering i bilden att göra. Detta beslut var menat att lägga fokus på själva bilden som blir redigerad och skapa ett gränssnitt som inte känns rörigt. Fokus har lagts på att de funktioner som är relevanta för stunden alltid ska finnas nära till hands, men vissa funktioner tar fler steg att hitta i programmet. Detta skulle kunna göra processen mindre effektiv till en början för vissa användare, men efter en tids användning skulle detta problem minska.

Målgruppen för programmet har försökts hållas så bred som möjligt under processen. Samtidigt har en viss datorvana förväntats från användarna för att förstå vissa begrepp och funktioner som exempelvis **gaussisk oskärpa** samt hur man navigerar i bilden. En person med stor vana av mer avancerade bildprogram skulle inte heller känna sig för begränsad i användningen av programmet, om personen hade realistiska krav på programmets funktion och syfte.

5.4 Utökningar och förbättringar

En utökning av programmet kan vara att lägga till fler filter. Detta hade ökat programmets användningspotential men även kunnat motverka själva designfilosofin med fokus på låg inlärningskurva. Därför skulle fler omstruktureringar och större designändringar behöva ske för att inte enkelheten skulle behöva offras. Ur en implementationsynpunkt är det alltså väldigt enkelt att lägga till mängder av funktioner, men det betyder inte att det är bra för användaren i slutändan.

En förbättring kan vara optimeringar kring hur bilddatan lagras och hur lagerinformationen hanteras. Stora bilder kan göra att programmet uppför sig långsammare då även beräkningarna blir i större skala. Ett sätt att göra det här skulle vara att göra flertrådad uppmålning av bilden samt att utnyttja flertrådighet på fler ställen i koden.

Andra kodförbättringar som skulle kunna göras är minnesoptimeringar. Med hjälp av komprimeringsteknik skulle programmets minnesanvänding kunna optimeras. Det hade kunnat tillåta både snabbare användning av programmet samt möjlighet för fler och mer kraftfulla funktioner.

5.5 Slutsats

I slutändan är Layers ett program som fungerar väl för det tänkta syftet - att vara ett enkelt bildredigeringsprogram. Det har ett passade gränssnitt som hjälper till att stärka syftet, samt en lagom mängd funktioner som erbjuds till användaren och möjlighet för utökning. Programmet hade kunnat effektiviserats och optimerats, speciellt när det kommer till hastigheten och minneshanteringen, men i nuvarande form är det en bra grund för ett ännu mer välfungerande program.

Referenser

[1] S Bakhshi,D Shamma,L Kennedy,E Gilbert, Åhy We Filter Our Photos and How It Impacts Engagement". 2015. Tillgänglig: http://comp.social.gatech.edu/papers/icwsm15.why.bakhshi.pdf (Hämtad 2017-04-19)

[2] E Evans "Domain-Driven Design Reference". 2015. Tillgänglig: http://domainlanguage.com/wp-content/uploads/2016/05/DDD_Reference_2015-03.pdf (hämtad 2017-05-05)

[3] Brett D Roads "Domain-Driven Design". University of Colorado Boulder, 2012 Tillgänglig: https://www.cs.colorado.edu/~kena/classes/5448/f12/presentation-materials/roads.pdf (Hämtad 2017-04-03)

[4] S. Walther, The Evolution of MVC — Stephen Walther", Stephenwalther.com, 2008. Tillgänglig: http://stephenwalther.com/archive/2008/08/24/the-evolution-of-mvc. (Hämtad: 2017-04-13).

[5] D. J. Skrien, "Object-oriented design using Java", 1st ed. New York, NY: McGraw-Hill Higher Education, 2009.

[6] J Ludwig Image Convolutions web.pdx.edu/~jduh/courses/Archive/geog481w07/Students/Ludwig_ImageConvolution.pdf) (Hämtad 2017-04-19)

[7] MAO. Vasilescu. "Image Filtering & Edge Detection". Stony Brook University, 2009. http://alumni.media.mit.edu/~maov/classes/vision09/lect/09_Image_Filtering_Edge_Detection_09.pdf (Hämtad 2017-03-28)

[8] Scrum.org, "What is scrum?". Tillgänglig: https://www.scrum.org/resources/what-is-scrum (Hämtad 2017-05-20)

[9] M Resnick,B Myers,K Nakakoji,B Shneiderman,R Pausch,T Selker, M Eisenberg Design Principles for Tools to Support Creative Thinking". 2005. Tillgänglig: https://pdfs.semanticscholar.org/7f7b/8ea2046454322608e2c021691e9d3a67f659.pdf (Hämtad 2017-04-19)

Bilagor

Akronymer

API Application Programming Interface. 8

gränssnitt Graphical User Interface. 1

HMVC Hierarchical model–view–controller. 2

MVC Model-View-Controller. 2, 3

MVP Model-View-Presenter. 2

MVVM Model-View-View-Model. 2

RGB Red-Green-Blue. 1

STAN Structure Analysis. 7

Ordlista

center stage designmönster som bygger på att en stor yta är fokus i mitten av ett program. 5

desktopformat det format som program följer på datorer. 1

easy extensibility en princip inom programmering som berättar hur man gör programvaran lätt att utöka. 4

flow tillstånd när en användare är helt inne i vad hen gör. 8

gaussisk oskärpa typ av oskärpa som ger högre viktning åt de pixlar som ligger närmare mitten. 9

gränssnitt den delen av ett program som användaren kan interagera med. 7

handler har ansvar över ett givet område. 8

interface ett sätt att i programmering strukturera upp klasser som har liknande funktionalitet. 7

Java-applikation program programmerat i Java för datorer. 1

JavaFX paket i Java som gör det enklare att skapa grafiska program. 1

keybord shortcuts ofta en kombination av tagenter som ett program kan känna igen och användaren kan använda för att snabbt utföra funktioner. 1

linjärt beroende när ett program är beroende av varandra i en linje och inte skapar cirklar. 7

menyfält navigationsverktyg som ger tillgång till programmets samtliga funktioner . 1

ramverk ramverk som beskriver hur något ska göras. 4

slider reglage som låter användaren välja mellan ett största och minsta värde. 5

systemarkitektur den design som koden i ett program följer. 4

toolbar används ofta för att presentera en rad funktioner och verktyg för användaren. 1