Tool Support for Kanban Boards in Software Development

Nawar Al-Mubaraki s062412

Kongens Lyngby 2011
IMM-M.Sc.-2011-61

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

In modern software development, it became a common practice to visualize the development process using whiteboards. Kanban is a visualization tool that is used to control the execution of a software development process. Kanban is used in modern development processes to visualize and control, what, when and how much to produce.

Kanban consists mainly of a whiteboard and set of cards. The whiteboard contains drawings and cards, where drawings visualize development phases, and cards represent userstories that can be attached to development phases.

The aim of this thesis is to develop a tool support for Kanban boards in software development. The tool simulates a whiteboard, that supports drawing, adding and characterizing cards. A number of features will be taken into account, that makes this tool unique compared to other Kanban and drawing tools.

The whiteboard is scalable, in a way such that whiteboard is scaled suitably to different monitor resolutions, without losing the overview of the whiteboard. That means, drawings and cards are scaled too, based on the size of the whiteboard. Furthermore, cards and drawings can be grouped together, by attaching cards to drawings. Cards are extended to represent more than just a userstory. By supporting card-hierarchy, a card can represent a new whiteboard.

Furthermore, various default features will be added as well, such like saving and loading the whiteboard, undo and redo actions, moving and deleting objects etc.

Resumé

I moderne softwareudvikling er det blevet almindelig praksis, at visualisere udviklingsprocessen vha. whiteboardtavle. Kanban er et visualiseringsværktøj, som anvendes til at kontrollere gennemførelsen af software udviklingsprocesser. Kanban anvendes i moderne udviklingsprocesser for at visualisere og kontrollere hvad, hvornår og hvor meget der skal produceres.

Kanban består hovedsagligt af en whiteboardtavle og et sæt af kort. Whiteboardtavlen indeholder tegninger og kort, hvor tegninger visualiserer udviklingsfaser, og Kanban kort repræsenterer brugerhistorier, som kan tilknyttes udviklingsfaser.

Målsætning med specialet er at udvikle et værktøj der understøtter Kanban tavler i softwareudvikling. Værktøjet simulerer en whiteboardtavle, som understøtter at tegne samt at indsætte og karakterisere kort. Der tages højde for en række udvidelser, som gør værktøjet unikt i forhold til andre Kanban eller tegne værktøjer. Tavlen skal være skalerbar således, at tavlens størrelse skaleres passende til de forskellige skærmopløsninger, uden at man mister overblikket. Dette indebærer, at tavlens indhold, tegninger og kort, bliver skaleret efter tavlens størrelse. Kort og tegning kan grupperes ved at tilknytte kort til tegning. Kort udvides til mere end blot en brugerhistorie. Ved at understøtte kort-hierarki, kan et kort også repræsentere en ny tavle.

Derudover tilføjes en række standard funktionaliteter, såsom at gemme og indlæse tavlens tilstande, fortryde eller gentage en handling, flytte eller slette objekter osv.

Forord

Dette kandidatspeciale er lavet ved DTU Informatik institut for Informatik og Matematisk Modellering, Danmarks Tekniske Universitet, som afsluttende projekt, for at opnå Civilingeniør titlen.

Specialet omhandler implementering af et værktøj der understøtter Kanban tavler i softwareudvikling. Specialet undersøger de forskellige problemstillinger og løsninger for at udvikle et unikt og intuitivt værktøj. Specialet startede i marts 2011 og blev afsluttet 15. august.

Acknowledgements

I would like to thank my supervisor, Associate Professor Hubert Baumeister at DTU's IMM, for his help and inspiration during this project. I would also like to thank my family, and especially my brother Sevan, for the support and the motivation. I thank my friends Allan Johnsen, Lars Frydendal, Rami Tayih and Amar Al-Soudi for participating in the usability-test and reading my report.

Indhold

Figurer

Tabeller

Listings

Kapitel 1

Introduktion

Whiteboard er så udbredt i dag, at det ses næsten overalt, i skoler, universiteter, arbejdspladser osv. Whiteboardtavlen tillader at skrive, tegne og visualisere de data man vil udveksle og kommunikere med. Whiteboardtavlen kan også bruges i sammenspil med post-it kort, til at planlægge og udføre opgaver. Undersøgelser [1] viser at det er vigtigt at kunne skitserer fri-hånds-skitsering, på en hurtig og nem måde. Det er på på baggrund af, at skitsering tillader folk at eksperimentere og udforske mulighederne, da idéer i begyndelsen ikke opstår velformede. Skitsering giver folk muligheden for at udtrykke sig, og igennem diskussion, udforme ideen. Andre undersøgelser [2] viser hvordan post-it kort bruges til at repræsentere informationer, som folk udveksler, ved at binde disse kort til et specifikt område i whiteboardtavlen. Whiteboardtavlen anvendes i utallige sammenhæng, såsom undervisning, brainstorming, udvikling osv. En af de måder, hvor whiteboardtavlen anvendes i udvikling og produktion, er Kanban.

Kanban er et japansk ord, som betyder tegnetavle eller reklametavle [3]. Begrebet blev brugt til at betegne et metaltegn eller et segl. I slutningen af 1950, begyndte Toyota at undersøge supermarkeder, og hvordan deres kunder fik hvad de ville, fik det når behøvede det og fik det i den rette mængde. Toyotas motivation var at anvende denne proces på fabrikken således, at kundens efterspørgsel sendes til lageret for at skaffe de krævede komponenter. Lageret fyldes efterfølgende efter behov, ligesom i et supermarked [4]. Det er Taiichi Ohno som udviklede Kanban, for at kontrollere produktionen og for at implementere Just-

in-Time (JIT) fremstilling hos Toyota. Det er først i 1970'erne Kanban begyndte at udbrede sig globalt, da det viste sig at metoden kan bruges til at minimere arbejde i processen mellem de forskellige udviklingsfaser.

Kanban er derfor baseret på "pull"-konceptet, hvor produktionen bestemmes i henhold til de aktuelle efterspørgsel fra kunder. Kanban-kort er et vigtigt element i Kanban, hvor der gøres brug af kort til at signalere efterspørgslen på materialer, som så flyttes fra lageret til produktionen. Kanban-kort kan betragtes, som en besked om at udtømme eller genopfylde et produkt.

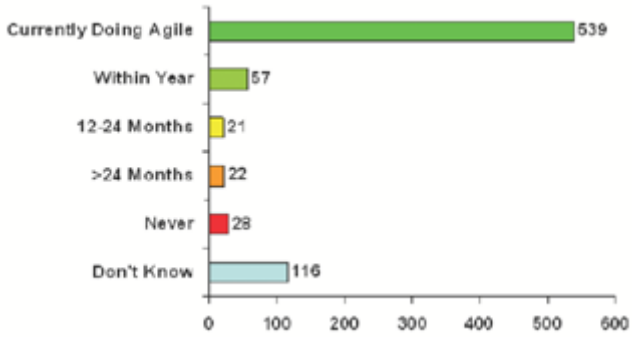
Kanban tavlen bruges til at tegne forskellige produktionsfaser og kortene bruges til at signalere, at en del af produktionen skal igennem en ny fase. Dette betyder at Kanban er en måde at udnytte whiteboardtavlen og post-it kort.

Et simpelt eksempel på anvendelse af Kanban, er tre-kurve-systemet. Systemet består af tre kurve, den første befinder sig i fabriksgulvet, den anden i lageret og den tredje hos leverandøren. Kanban kort bruges til at signalere, når et produkt skal flyttes fra det ene til det anden kurv. Når produktionsdele, i den første kurv, nærmer sig udtømning, fyldes den igen, og det samme gælder lageret og leverandøren. Et andet eksempel kan være, et varehus, af enhver type, med et tilhørende lager, som indeholder et antal af forskellige varer. Når kunder køber, hvad de har brug for i varehuset, sendes der et signal til lageret, om hvilke varer der blev solgt. Dette indebærer, at lageret får besked, om hvilke vare der er ved at blive udsolgt, og hermed sendes genopfyldning til varehuset. Det samme princip, som nævnt i det sidste eksempel, kan anvendes hos supermarkeder, bagerier og i mange andre sammenhænge.

Kanban tavle bliver brugt mere og mere i moderne udviklingsprocesser, som et visualiseringsværktøj, der hjælper med at give et overblik over, hvad er skal produceres, hvornår det skal produceres og hvor meget der skal produceres. Da Kanban har forskellige definitioner, er det vigtigt at understrege, at Kanban ifølge dette speciale betragtes som et visuel proces management system og ikke som udviklingsproces. Det betyder, at vi ser Kanban som et værktøj og ikke som en metode.

En undersøgelse fra 2007, Agile Adoption Survey [5] hvor mere end 700 peroner blev spurgt, viser at agile softwareudvikling er så udbredt, at næsten 70% af virksomhederne allerede anvender agile for et eller flere projekter, og knap 7% kommer til at anvende agile i løbet af et år. Disse resultater stammer fra 2007 og det er forventet at agile i softwareudvikling er vokset siden da. Figure 1.1 på følgende side viser undersøgelsen resltater.

Med denne baggrund, er der også voksende behov for værktøjer, der visualisere projektets tilstand, hvor en af mulighederne er Kanban tavlen. Kanban tavlen



Figur 1.1: Agile anvendelse

bliver tit brugt i forbindelse med softwareudvikling for at visualisere udviklingsprocesser, hvor tavlen indeholder de forskellige tegninger der repræsenterer forskellige udviklingsfaser, og kort repræsenterer opgaver tilknyttet til en udviklingsfase.

Figur 1.2 viser en klassisk Kanban tavle, hvor tavlen er opdelt i forskellige udviklingsfaser, og kort er tilknyttet til faserne. Kort flyttes igennem udvikling, for eksempel når et kort er færdiganalyseret, flyttes det til design, implementering, test osv.

Figur 1.2: Et eksempel på en simpel Kanban tavle

Kanban kan repræsenteres på forskellige måder, hvor tegninger kan være af forskellige former, på vilkårlige rækkefølge, og kort kan have forskellige farve og indhold. Formen samt tegninger og kort kan ændrer sig over tiden og fra projekt til andet.

Figur 1.3 viser en lidt mere kompliceret Kanban tavle, der indeholder anderledes udviklingsfaser. Tavlen indeholder også forskellige kort som hver især har sin egen karakteristiske farve. Hver farve repræsenterer en type, som kan være bug, feature, note osv.

Figur 1.3: Et eksempel på en avancerede Kanban tavle

Figur 1.4 viser endnu et eksempel på en Kanban tavle, som er simplere end den forrige figur. Disse forskellige eksempler viser at Kanban tavler ikke har en fast form, og hvert projekt kan repræsenteres forskelligt, og kan ændres over tiden.

Figur 1.4: Et andet eksempel på en avancerede Kanban tavle

1.1 Problembeskrivelse

Whiteboardtavler har ingen prædefineret fast form, og kan bruges som Kanban tavler, eller som mange andre ting. Dette indebærer også, at Kanban tavlen kan præsenteres på forskellige måder som vist før, og dermed er man ikke bundet til en specifik repræsentation. Motivationen er at udvikle et værktøj der simulerer en whiteboardtavle med post-it kort, der kan brugs som Kanban tavle, men også i andre sammenhæng. Værktøjet skal kunne simulere et generelt whiteboardtavle, som kan tilpasses til at repræsentere forskellige Kanban tavler, såsom i de ovenstående figurer.

Værktøjet skal naturligvis understøtte fri-hånd-tegning, hvor forskellige tegningsformer og farver er tilgængelige og kan frit anvendes. Kort kan indsættes og tilknyttes forskellige tegninger, samt baggrundsfarven og indholdet kan redigeres.

Der findes også mange forskellige størrelser af skærme, bærbare og notebooks, og det betyder, at der er brug for at vise tavlen i de forskellige skærmeopløsninger uden at miste overblikket.

Kort er også et nøglekomponent i kanban og det er endda muligt at gøre det endnu mere fleksible, ved at tillade kort at være ikke blot en brugerhistorie, men også en hel nye tavle. Det betyder at der er tale om hierarki struktur, hvor kort kan også repræsentere en Kanban tavle.

Kanban tavler anvendes ikke kun specifikt for softwareudvikling men også i forbindelse med produktion af enhver art, og derfor kan værktøjet anvendes ikke kun i forbindelse med softwareudvikling. Whiteboardtavlen og kort kan generelt også bruges i andre sammenhænge end Kanban, og derfor er der ingen begrænsning med hensyn til hvad værktøjet skal bruges til.

På trods af, at der allerede findes digitale kanban værktøjer, opfylder de ikke whiteboardtavlens krav, da de ikke understøtter mulighed for at tegne på tavlen. Da whiteboardtavlen er en nøglekomponent i kanban er det vigtigt at have den med i værktøjet, uden at miste de andre funktionaliteter. Under analyse afsnittet, undersøges de forskellige kanban værktøjer, for at identificere mangel på funktionalitet. Disse funktioner skal kombineres til et unikt værktøj, som letter arbejdet end andre værktøjer. Det betyder naturligvis, at hensigten er at lave et nyt og unikt værktøj, som ikke allerede findes. Afsnittet relateret arbejde vil beskrive de forskellige værktøjer i detaljer og vise hvorfor disse værktøjer ikke opfylder behovet.

1.2 Rapport struktur

Dette afsnit vil ganske kort beskrive projektets indhold og udpege hvilket kapitel der indeholder hvilken del af rapporten.

Kapitel 2 dokumenterer projektets planlægning og anvendte planlægnings metoder.

Kapitel 3 analyserer og definerer problemet ved at kigge i de allerede eksisterende løsninger.

Kapitel 4 beskriver hvordan værktøjet er designet og hvorfor det er designet på den måde

Kapitel 5 giver et implementerings overblik over hvordan tingene fungere hvordan kodens struktur ser ud.

Kapitel 6 tester og godkender funktionaliteterne vha. en manuel-, automatisk- og brugervenligheds test.

Kapitel 7 evaluerer projektet og hvordan det kan forbedres.

Kapitel 8 giver en afsluttende konklusion.

Kapitel 2

Planlægning

Da formålet er at udvikle et nyt værktøj med masser af nye funktionaliteter, er det vigtigt at planlægge udviklingen omhyggeligt. Vi valgte at anvende en agile metode med få ugers mellemrum mellem hver iteration, for at udvikle hurtigt og for at have en fortsat idé om projektets udvikling.

Planlægning fylder og betyder en del i dette speciale. Der valgtes at anvende extreme programming (XP) [6, 7] som udviklingsmetode, som uddybes i det følgende afsnit.

2.1 Udviklingsproces

Extreme programming er en relativ ny udviklingsproces, da det første XP projekt var startet i starten af 1996. Det var Kent Beck som skabte XP igennem sit arbejde med C3 [8] projektet, hvor han løbende prøvede på at forbedre udviklingsprocessen. Det viste sig at metoden var succesfuld og i 1999 skrev Kent Beck sin bog "Extreme Programming Explained". Derefter og i slutning af 1990'erne, fik XP meget omtale og blev anvendt i mange projekter. XP var og stadig er en succes, fordi det fremhæver kundens tilfredsstillelse ved at levere, hvad kunden har behov for så tidligt som muligt.

XP er en agile metode, som er åben for ændringer og tilpasninger løbende i dens proces. XP består af en kort timeboxing [9], som er en planlægningsteknik der anvendes til at dele projekts forløb i små perioder (normalt mellem 2 til 6 uger). Hver periode har sin frist, hvor brugerhistorier bliver designet, implementeret og testet. Det er kunden som definerer prioritering af hvilke brugerhistorier der skal udvikles først. Det er også kunden, der godkender om det der blev udviklet, faktisk var det der var brug for.

XP er beregnet til gruppearbejde, og derfor i nogle områder, har det været lidt udfordrende at udnytte eller benytte XP koncepter.

Brugerhistorier tjener det samme formål som use cases, men de er ikke det samme, for de bliver brugt for at estimere planlægnings varighed. De bruges også i stedet for et langt kravspecifikation dokument. Brugerhistorierne kan blive skrevet af kunder, som det funktionalitet systemet skal kunne, i form af få ikke-tekniske sætninger. Forskellen mellem brugerhistorie og kravspecifikation er, at brugerhistorien giver nok detaljer til at estimere, hvor lang tid det minimum kræver for at udvikle den. Når en programmør når til implementering af en specifik brugerhistorie, hentes detaljerne og specifikationerne. Det kan også gøres den anden vej rundt, således at kunden definerer kravspecifikation og udviklerne genererer brugerhistorierne.

Dette speciale mangler en real kunde, men min vejleder, Hubert Baumeister, blev betragtet som kunde og har taget denne rolle for at bestemme og definere hvilke brugerhistorier der skal udvikles.

I starten af specialet, blev der defineret en lang liste af brugerhistorier, og nye brugerhistorier blev også tilføjet løbende i udviklingsperioden. Brugerhistorierne blev prioriteret og identificeret mht. sværhedsgrad således, at der udvikles et antal af brugerhistorierne i en timeboxing af ca. 2-3 uger. Da jeg selv var med til at definere og diksutere brugerhistorierne, var der ikke behov for flere detaljer og specifikationer, for dem kunne jeg selv angive. Planlægning af hver iteration skete ved en personlig samtale med min vejleder, for at planlægge hvordan den næste iteration skulle forgå og hvilke brugerhistorier skulle implementeres.

Hver iteration går igennem en designfase, som består af at designe brugerhistorierne hver for sig. Design i XP skal være så simpelt som muligt, da det altid er nemmere og hurtigere. Da det er subjektivt at definere simpelheden, er det gruppen, som definerer, hvad et simpelt design er. Dette har været udfordrende da jeg er alene om designet, og det har betydet at jeg har designet systemet så simpelt som muligt, baseret på litteraturer, erfaring og rådgivning fra min vejleder. XP opfordrer udviklere til at undgå at designe ekstra funktionaliteter, som måske kan bruges senere, for undersøgelser viser at 90% af de ekstra design bliver aldrig brugt [10]. Dette blev overholdt i designet, og kun de funktionaliteter

der var brug for, blev designet.

Hver iteration går igennem en implementeringsfase, som består af, at fortolke design til fungerende kode. I XP anvendes parprogrammering til at producere og udgive fungerende kode. Parprogrammering er en softwareudviklingsteknik, hvor to programmører koder sammen i én computer. Den ene programmør koder mens den anden observerer og gennemgår koden samtidigt. Personen der skriver kode kaldes Driver og den der observerer kaldes Observer, og programmørerne skiftes om de to roller jævnligt. Mens Driver koder, overvejer Observer forbedringer og rettelser til koden. Denne teknik har ikke vært muligt at benytte, da projektet består af en enkel programmør.

Udvikling i XP er test-drevet, som betyder at testen kodes før koden. Der argumenteres for, at det er hurtigere at generere kode, når testen er skrevet først. Denne teknik har vært besværlig at anvende, og det skyldes, at jeg ikke kunne benytte den til at implementere koden hurtigere. I stedet endte jeg i de fleste tilfælde med at skrive koden og testen samtidigt.

Ligesom de andre faser, går hver iteration igennem en test fase, hvor systemet testes og godkendes for at være klar til udgivelse. Test består af automatiseret unit-test og acceptance test, hvor hver især bliver testet i hver iteration, i forhold til de implementerede brugerhistorier. Denne teknik blev også overholdt og viste sig at være en succes sammenlignet med vandfald metoden, hvor testen gennemføres i slutningen.

Generelt er jeg positiv indstillet over for at anvende XP som udviklingsmetode, på trods af at der er nogle koncepter (såsom parprogrammering), som var svære at benytte, pga. udvikling består af en enkel person. Resultatet af at anvende XP i dette speciale er også tilfredsstillende og forskellen mellem XP og vandfald metoden kan godt mærkes med fordel til den førstnævnte.

Udvikling bestod af 5 iterationer, hvor den første var på 4 uger. Den anden og fjerde iteration tog 2 uger hver, mens tredje og femte iteration tog 3 uger hver. Efter den femte iteration, blev værktøjet testet af min vejleder, som resulteret i en lang liste af rettelser. De fleste af disse rettelser blev rettet mens rapporten blev skrevet.

Kapitel 3

Analyse

Dette kapitel vil kigge på det relaterede arbejde og de tidligere forsøge for at lave et kanban værktøj. Der påpeges hvilke teknologier og værktøjer der skal anvendes for at udvikle dette speciale. Herefter defineres en fyldestgørende kravspecifikation, der dækker projektets omfang, og hvad der forventes at have i værktøjet.

3.1 Relateret arbejde

Dette afsnit vil præsentere andres anbefalinger og forsøg på at løse problemet. Der er lidt akademisk arbejde udført i dette område, hvorimod masser af kanban værktøjer i den kommercielle verden.

3.1.1 Akademisk

Frank, Robbert, Xin and Josyleuda fra University of Calegary har udarbejdet en rapport "Tools for Supporting Distributed Agile Project Planning", der identificerer og beskriver kravspecifikation for digitale agile værktøjer [11]. Rapporten

beskriver, hvilke funktionelle krav agile værktøjer skal kunne for at opnå agile planlægnings mål. Målet er at kontrollere udviklingsprocessen, starte ny udvikling og planlægge iterationer og forstærke samarbejdet mellem de forskellige udviklings grupper. Rapporten starter med at dokumenter de basale funktionaliteter, herunder at det skal være muligt at tilføje, redigere og slette planlægnings objekter, såsom kort, brugerhistorier, tegninger osv. Det skal også være muligt at flytte kort fra en udviklingsfase til en anden. Det er også vigtigt, at der gives en visuel karakteristik for kort, således at man kan se forskel på om dette kort er bug, feature eller blot en note. Disse krav virker lidt trivielle, da det er oplagt at værktøjet skal kunne tilføje, slette og redigere planlægnings objekter.

Derudover, beskriver rapporten vigtigheden af at kunne gemme ældre iterationer, for at hjælpe med at estimerer nye opgaver i fremtiden. På samme måde, skal det være muligt at planlægge adskillige iterationer ad gangen, for at give muligheden for at planlægge på lang sigt.

Autentificering ifølge rapporten er også et vigtigt aspekt, at planlægning kun må kun ændres af dem, der har adgang til det. Værktøjet skal også integreres i udviklingsmiljøet således, at når en ændring fortages i udviklingsmiljøet, kan værktøjet opdateres.

Udover planlægnings funktionalitet, beskriver rapporten andre funktionaliteter som understøtter samarbejdende interaktion mellem udviklings grupper. For eksempel, hvis værktøjet skal bruges som et distribueret system, skal det være muligt for alle medlemmer at redigere i real-time. Ændringer skal sendes videre til alle deltager således, at de er i stand til at redigere i værktøjet samtidigt. Rapporten beskriver yderligere, at når en ændring fortages i værktøjet, skal projektets medlemmer notificeres.

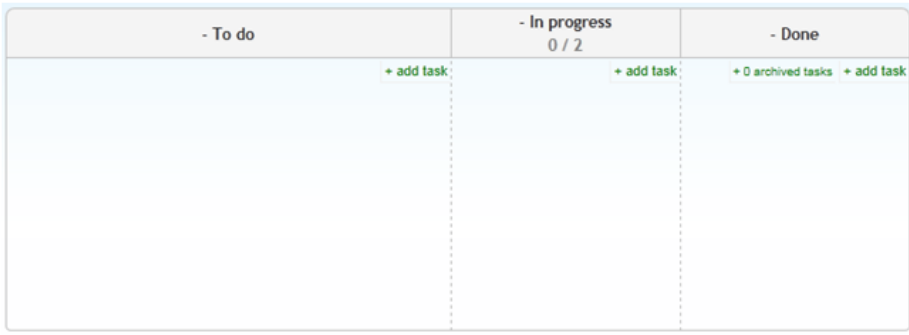
3.1.2 Kommercielt

Der findes mange kanban værktøjer, både web- og desktopbaserede. Værktøjerne varier mellem at være gratis eller dyre, og simple eller komplicerede. Derfor blev der udvalgt nogle værktøjer, som indeholder flest funktionaliteter.

De følgende webbaserede værktøjer blev udvalgt for at undersøge og identificere de funktionelle krav, som de har tilfældes. Valget er baseret på funktionaliteten og tilgængeligheden, da de følgende værktøjer kan afprøves gratis.

3.1.2.1 KanbanTool

KanbanTool er en webbaseret kanban værktøj, som er tilgængelig via www.kanbantool.com. Værktøjet kan afprøves gratis i 30 dage, hvor man kan oprette en enkel kanban tavle. Tavlen består af rækker og koloner, som hver repræsenterer en udviklingsfase. Når en tavle oprettes, fås en mulighed at anvende standard skabelon som består af 3 koloner, To do, In progress og Done, som kan ses i figur 3.1.



Figur 3.1: Kanban Tool tavlen

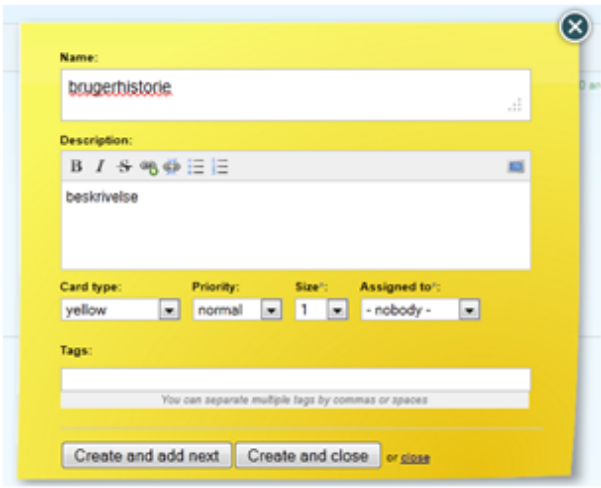
Inden tavlen oprettes, er der en mulighed for at skræddersy tavlen med ekstra rækker og koloner, men dette kan kun gøres før tavlen oprettes og når tavlen er oprettet er det ikke muligt at ændre senere.

Kort kan tilføjes og redigeres, ved at indsætte titel, beskrivelse, prioritet, størrelse og type, som kan ses på figur 3.2 på næste side. Beskrivelsen kan formateres vha. *Rich-Text-Editor*. Prioriteten består af et tal som definerer hvor vigtig kortet er. Typen kan vælges ud fra prædefinerede typer, som hver har sin egen farve. Kortets baggrundsfarve kan være en af fem valgmuligheder. Kort kan tilknyttes til en kolonne og kan tildeles en bruger, som er ansvarlig for det.

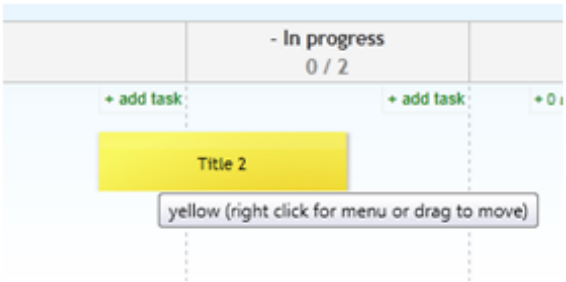
Kort kan flyttes frem og tilbage mellem kolonerne vha. træk-og-slip funktion, som kan ses på figur 3.3 på følgende side. Kortet kan kun placeres i midten af kolonnen, og det er ikke muligt at placeres i en vilkårlig position i tavlen.

Kort kan redigeres og slettes ved at højreklikke, som vises på figur 3.4 på næste side. Redigering samt sletning kan ikke fortrydes, da værktøjet mangler en fortrydelses funktion i det hele taget.

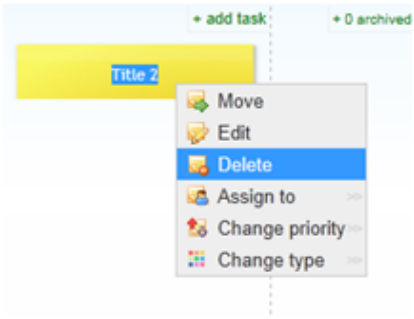
Derudover har værktøjet analytisk funktion, som kan bruges til at vise en graf over, hvor mange kort der er, hvor mange er under udvikling, hvor mange er færdige osv. Figur 3.5 på side 15 viser en graf over korts fordeling.



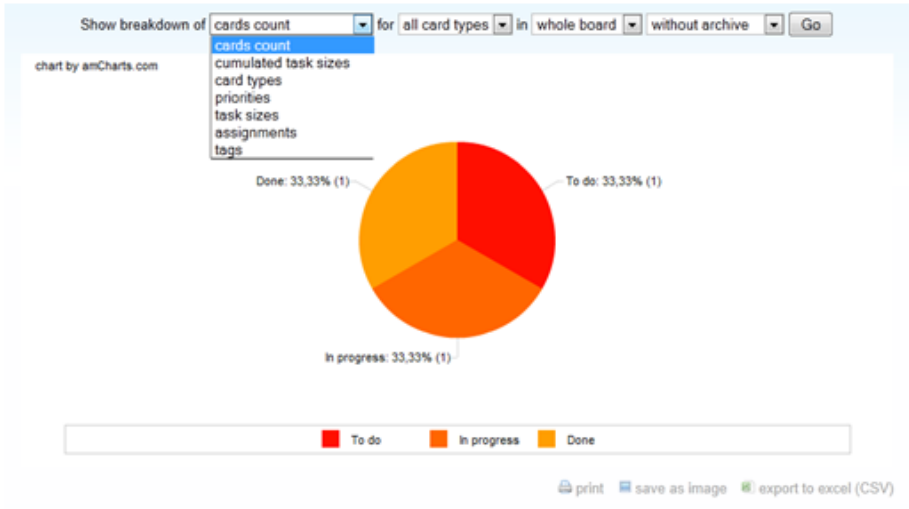
Figur 3.2: Oprettelse af kort i Kanban Tool



Figur 3.3: Flytning af kort i Kanban Tool



Figur 3.4: kort redigering i Kanban Tool



Figur 3.5: Analytiske funktioner i Kanban Tool

3.1.2.2 Kanbanery

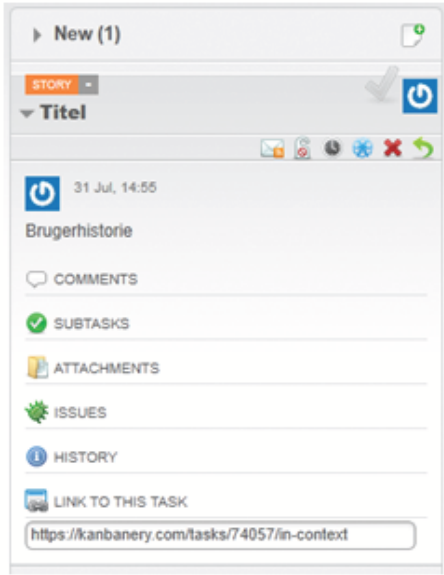
Kanbanery er en webbaseret kanban værktøj, som er tilgængelig via www.kanbanery.com. Værktøjet kan anvendes gratis med en enkel tavle og maksimum 2 brugere. Værktøjet består af 3 koloner, New, In progress og Done, som vises på figur 3.6. Disse koloner kan ikke redigeres, og nye koloner kan ikke tilføjes.

Figur 3.6: Tavlen i Kanbanery

Kort kan tilføjes med titel, prioritet, type, estimat og beskrivelse, som kan ses på figur 3.7 på følgende side. Beskrivelsen er en simpel ikke-formateret tekst. Kort type kan være en af tre muligheder, Story, Bug eller Chore, mens baggrundsfarven ikke kan ændres. Kort kan flyttes mellem de 3 koloner vha. simpel træk-og-slip.

Der kan tilføjes kommentarer, vedhæftes filer og oprettes underopgaver til kort. Kort kan slettes og deres frist kan defineres. Der er også en redigeringshistorie som viser, hvordan kort er blevet behandlet. Dette kan ses på figure 3.8 på næste side.

Figur 3.7: Oprettelse af kort i Kanbanery



Figur 3.8: Kort funktioner i Kanbanery

3.1.2.3 LeanKitKanban

LeanKitKanban er en webbaseret kanban værktøj, som er tilgængelig via www.leankitkanban.com. Som standard består tavlen af 3 koloner, som kan ses på figur 3.9, men det er muligt at tilføje eller slette koloner undervejs.

Figur 3.9: Tavlen i LeanKitKanban

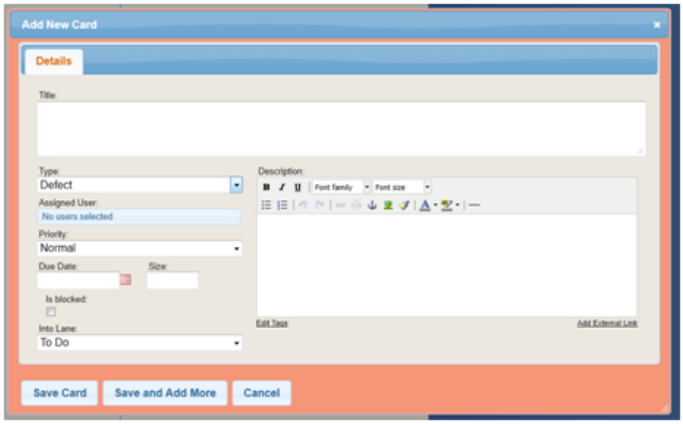
Kort kan tilføjes med titel, beskrivelse, type, prioritet, dato og størrelse. Beskrivelsen formateres vha. Rich-Text-Editor, hvor bl.a. billeder og link kan indsættes. Prioriteten kan være Low, Normal, High eller Critical. Typen kan være Defect, Feature, Improvement eller Task, og hver type har sin egen prædefineret farve. Dette kan ses på figur 3.10 på næste side

Kort kan flyttes mellem kolonerne via. træk-og-slip. Korts position defineres i forhold til kolonnen og hvor mange kort der er, og det betyder at kort ikke kan placeres et vilkårligt sted på tavlen. Kort kan slettes og arkiveres i arkiv og backlog. Værktøjet har hverken analytiske funktioner eller bruger administrering.

3.1.2.4 Kanban Pad

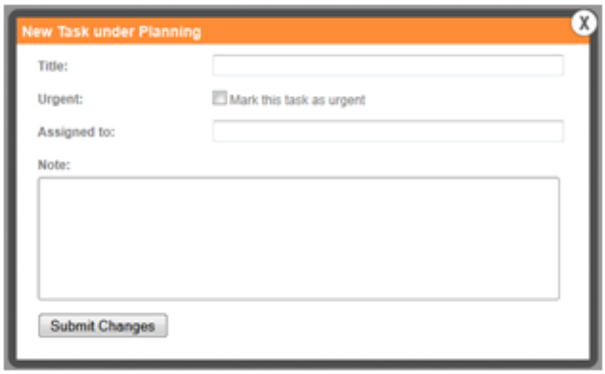
Kanban Pad er en webbaseret kanban værktøj, som er tilgængelig via www.kanbanpad.com. Som standard består tavlen af 5 koloner, som vises på figur 3.11 på følgende side, hvor nogle af dem kan slettes, men nye kan ikke tilføjes.

Kort kan tilføjes med titel og beskrivelse, og der kan defineres hvilken bruger som er ansvarlig, som kan ses på figur 3.12 på næste side. Det er ikke muligt at ændre baggrundsfarve, eller formatere beskrivelsen. Kort kan slettes, og flyttes på samme måde som de andre værktøjer, vha. træk-og-slip mekanismen.



Figur 3.10: Oprettelse af kort i LeanKitKanban

Figur 3.11: Tavlen i Kanban Pad



Figur 3.12: Oprettelse af kort i Kanban Pad

3.1.3 Mangel på funktionalitet

Alle værktøjerne mangler en tegnetavle der understøtter at tegne, som ifølge undersøgelser [1] er en essentiel del af whiteboardtavlen. Ingen af værktøjerne understøtter omsortering af udviklingsfaser. Det er ikke muligt at skalere kanban værktøjer samt dens udviklingsfaser og kort. Kort kan kun indeholde en brugerhistorie, som begrænser hvad kortet kan repræsentere. Derudover, mangler der også en fortrydelses og gentagelses funktion, som hjælper brugeren med at rette en handling. Det er heller ikke muligt at gemme forskellige kopier af projektet, for at indlæse dem senere.

3.2 Funktionel kravspecifikation

Afsnittet beskriver de vigtigste funktionelle krav, der opstilles til værktøjet. Disse krav skal definere, hvordan værktøjet skal fungere, og hvad det skal kunne. Kravspecifikationen beskriver generelt de krav, som systemet skal have i uformel form, som senere kan bruges til at definere brugerhistorier.

De tekniske krav:

- Det skal være muligt, at gemme kanban tavlen og hente det på et andet givent tidspunkt.
- Der skal være en fortryd og gentag funktion, for at fortryde eller gentage en handling. Dette vil være nyttigt i situationer, hvor brugeren ved en fejl har udført en uønsket handling.
- Der skal være visuel feedback på valg, flytning og størrelsestilpasning af tavlens objekter.

Whiteboardtavlen:

- Det skal være muligt at tegne på whiteboardtavlen med forskellige tegningsformer og farver.
- Værktøjet skal kunne repræsentere whiteboardtavlen i forskellige skærmeopløsninger ved at skalere tavlen samt dens tegninger og kort.
- Tegninger på tavlen skal være flytbare.
- Tegninger skal kunne slettes.

Kort:

- Det skal være muligt at tilføje og slette kort.
- Kort skal kunne flyttes, ligesom tegninger.
- Det skal være muligt at tilknytte kort med tegninger.
- Når et kort er tilknyttet en tegning, skal kortet følge med når tegningen flyttes.
- Korts indhold skal kunne tilføjes og redigeres.
- Der skal være kort-hierarki således, at et kort kan repræsentere en hel ny kanban tavle.
- Kort type skal kunne karakteriseres ved at ændre kortets baggrundsfarve.
- Det skal være muligt at tilpasse størrelsen på kort.

3.3 Ikke-funktionel kravspecifikation

- Værktøjet skal udvikles i Java.
- Swing grafiske komponenter skal bruges frem for AWT.
- Værktøjet kræver ikke programmeringserfaring for at anvende.
- Der skal tage hensyn til brugervenlighed.

3.4 Brugerhistorier

Ud fra kravspecifikation, defineres brugerhistorierne i form af små opgaver, som kan ses på tabel 3.1 på følgende side. Nye brugerhistorier blev tilføjet, og prioriteret løbende i udviklingsprocessen. Dette blev udført i starten af hver iteration, som varede 2-3 uger (udover iteration 1 som tog 4 uger), i en periode der strakte sig over 4 måneder.

Rækkefølgen definerer prioriteten af brugerhistorierne, og antallet af brugerhistorier som blev udviklet hver anden uge er afhængig af sværhedsgrad.

Tabel 3.1: Brugerhistorier

Titel	Brugerhistorie
1. iteration	
Tegne på tavle	Brugeren kan oprette et nyt tegningslærred, som kan tegns på
Flytte tegninger	Brugeren kan flytte tegningslærred rundt på tavlen
Tilføje kort	Brugeren kan tilføje nye kort
Flytte kort	Brugeren kan flytte kort rundt i tavlen
Kort følge med tegningslærred	Brugeren kan flytte en tegninger som beskrevet før, dette vil betyde at alle kort, der er tilknyttet tegningen, vil følge med
Brugerhistorie	Et kort kan indeholde en brugerhistorie
Kort indhold	Brugeren kan vise kortets indhold
At gemme	Brugeren kan gemme tavlens tilstand
At indlæse	Brugeren kan hente tavlens tilstand
2. iteration	
Understøttelse af forskellige tegnings former	Brugeren kan tegne med forskellige former, såsom linje, polylinje, firekant, ellipse og tekst.
Skalering	Brugeren kan skalere størrelsen af tavlen og dermed størrelsen af tegninger og kort.
3. iteration	
Automatisk tilpasning af tegningsstørrelse	Systemet tilpasser tegningslærred størrelse automatisk, efter man er færdig med at tegne.
Slette tegninger	Brugeren kan slette allerede eksisterende tegninger
Slette kort	Brugeren kan slette allerede eksisterende kort
4. iteration	
At tegne mens tavlens er skaleret	Brugeren kan tegne i forskellige tavle størrelser
Visuel feedback	Systemet skal kunne visualisere forskellige visuelle feedback baseret på udvælgelse, flytning og størrelsestilpasning.
Kort hierarki	Et kort kan repræsentere en ny tavle
5. iteration	
Korts baggrund	Brugeren kan ændre korts baggrund
Fortryde/gentage	Brugeren kan fortryde eller gentage en tegnings handling
Redigere teksten	Bruger kan redigere en allerede eksisterende tekst
Tegnings farver	Brugeren kan vælge at tegne med farver

3.5 Use Cases

Normalt erstatter brugerhistorier use cases i XP, da brugerhistorierne er selve definitionen på specifikationen. Use cases kan bruges analytisk til at generere flere brugerhistorier for at sikre at alle mulige scenarier er tiltænkt. Hver use case repræsenterer et unikt scenarie for en brugerhistorie, som hver især prioriteres uafhængigt. Use case beskriver en scenarie, som kan testes og dermed kontrolleres, om det dækker kundens behov.

Alle use cases forudsætter at man allerede har startet værktøjet. Use cases diagrammer findes under bilag A.

3.6 Teknologier og værktøjer

Dette afsnit giver en kort introduktion til de værktøjer og teknologier, som bliver brugt i dette speciale. Der gives kort forklaring for hver teknologi og værktøj i tilfældet af, at læseren ikke er bekendt med dem.

3.6.1 Java

Af erfaringsmæssige grunde, valgte jeg at programmere projektet i Java, idet der er mulighed for at understøtte flere platforme, såsom Windows, Unix, Mac OS, Android osv. Jeg fravalgte at implementere dette speciale som en eclipse-plugin, og det skyldes, at der ønskes, at værktøjet virker uafhængigt i forskellige platforme uden at kræve eclipse at være installeret. Derudover er eclipse-plugin udvikling en proces for sig selv, og jeg har tidligere haft en dårlig erfaring med udvikling af flere eclipse-plugin, hvor tingene ikke kunne fungere, som det skulle.

3.6.2 Swing

Swing er den primær Java grafiske toolkit, som er en del af Java Foundation Classes (JFC)[12]. Swing var i starten en tilføjelse til Java-standarden, som indeholdte Abstract Window Toolkit (AWT)[13]. AWT's udseende er platformafhængig, som det har betydet, at programmets udseende ser væsentligt anderledes ud end det, der er skrevet direkte til det pågældende operativsystem. Swing derimod er platformuafhængig, hvor standard Swing klasser understøtter

en række look & feel valg. Swing grafiske komponenter kan opføre sig som de tilsvarende native komponenter uden at skulle skrive platformafhængig kode, da look & feel systemet klar denne opgave. Komponenternes udseende kan tilpasses yderligere uden betydelige ændringer i kode.

Derfor vælges Swing fremover for AWT for at sikre et bedre udseende for værktøjet. Det er vigtigt at være konsekvent, når der vælges Swing komponenter, således at det ikke blandes med AWTs grafiske komponenter, da dette vil medføre grafiske problemer. AWT er *heavyweight* komponent, som er associeret med de komponenter, der tilhører det underliggende operativsystem. Dette kaldes også en peer komponent, at AWT snakker med den underliggende operativsystem, for at bestemme hvordan komponenten skal se ud. Swing er en *lightweight* [14] komponent, som er ikke associeret med operativsystemet og fremtoningen af komponenten, bestemmes af Java Virtual Machine. Bladning af Swing (*lightweight*) og AWT (*heavyweight*) komponenter betyder, at komponenter kan overlappe hinanden [15].

Det er dog ikke altid muligt at fuldstændigt undvære AWT, og det skyldes at der findes ikke-grafiske komponenter, såsom Event Listeners [16], der ikke findes i Swing og som essentiel for at udvikle desktop baseret værktøjer. Men så længe AWT bruges, når det er nødvendigt og ikke er i modstrid med Swing, burde det ikke give nogle grafiske problemer.

3.6.3 Eclipse

Eclipse er en af de mest kendte og anvendte IDE'er for Java programmer. Den indeholder en tekst editor samt en kompiler og en debugger. Eclipse indeholder også test-miljøet JUnit, som gør det nemt at opstille en automatiseret test.

3.7 Opsummering

Dette kapitel præsenterer en analyse af de tidligere forsøg og anbefalinger for at lave en digital kanban tavle. Ud fra analysen bestemmes nogle funktionelle og ikke-funktionelle krav til at definere projektets omfang. Der er også givet en kort introduktion til de forskellige teknologier og værktøjer som anvendes til at udvikle dette værktøj.

Kapitel 4

Design

Dette kapitel vil beskrive de valg og fravalg der blev truffet igennem designfaset. Værktøjets arkitektur vil blive skitseret og designvalget vil blive forklaret med begrundelse.

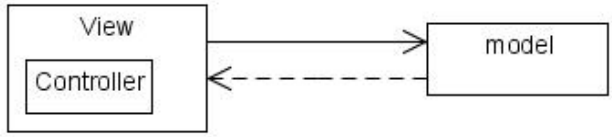
4.1 Design principper

Afsnittet indeholder de primære designprincipper og valg der er benyttet for at opnå et pænt og velstruktureret design.

4.1.1 Model, View & Controller

Som overordnet designprincip anvendes Model-View-Controller (MVC) [17] , så alt kode der håndterer tavlens tilstand (modellen) er afkoblet fra koden der håndterer grafikken (præsentationen). Dette princip vil lette arbejdet og opdeling mellem data og den grafiske præsentation. Modellen vil stå for data mens præsentationen vil stå for den grafiske repræsentation af data.

Figur 4.1 viser at præsentationen har en direkte associering med modellen, for at hente tilstanden, mens modellen har en indirekte associering med præsentationen igennem kontrolleren. Kontrolleren er en observatører, som registrerer brugerens input og sender det videre til modellen. Det betyder at brugeren interagerer med brugergrænsefladen, kontrollen behandler inputtet fra brugergrænsefladen, og notificerer modellen, som kan resultere i ændring af modellens tilstand. Når modellen ændres, vil brugergrænsefladen opdateres i forhold til det, da brugergrænsefladen har en direkte associering med modellen.



Figur 4.1: MVC diagram

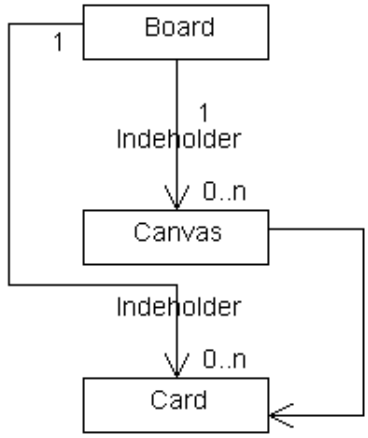
Opdeling har sine fordele, da det er lettere at forstå hvordan værktøjet fungerer hvis man ikke behøver at koncentrere sig om modellen og præsentationen samtidigt. At modellen er uafhængig af præsentationen gør, at man kan ændre brugergrænsefladen uden at det har konsekvenser for modellen. Kode der håndterer grafik og kode der håndterer tilstanden skrives ret forskelligt, og at have dem begge samlet et sted vil medføre at koden typisk bliver rodet, og mere besværlig at ændre.

4.2 Javabønner

De fleste af modellens klasser i værktøjet er javabønner [18]. En javabønne er en serialiserbar klasse hvor alle dens egenskaber skal kunne tilgås med *getters* & *setters*. Javabønner gør det nemmere at ændre objekternes egenskaber i en vilkårlig rækkefølge vha. *getters* & *setters*. Idéen bag at gemme objekters data som egenskaber, i stedet for værdier i felter er, dels at man frit kan omdøbe de underliggende værdier og dels at en *get* eller *set* metode ikke nødvendigvis blot sætter en enkelt variabel til en given værdi. Et eksempel på et problem man på den måde undgår, kunne f.eks. være hvis man beskriver en linje. Linjen kan både beskrives ud fra dens endepunkter, og ud fra dens startpunkt, størrelse og retning. Hvis på et tidspunkt menes, at det vil være smart at skifte den måde man repræsenterer linjen på, behøver man ikke at ændre filformatet, men der kan bare ændres linjens *getter* og *setter* metoder.

4.3 Domænemodel

Følgende domænemodel illustrerer overordnet hvordan systemet er designet og hvordan de forskellige dele er relateret til hinanden. Systemet består af 3 dele; *Board* som er tavle, *Canvas* som er tegningslærred og *Card* som er kort. En tavle indeholder et antal af tegningslærreder og kort. Systemets få dele letter designet og gøre det nemmere at håndtere implementering, da man ikke behøver at koordinere mellem flere objekter og instanser.



Figur 4.2: Domænemodel

Grunden til at det er tavlen og ikke tegningslærredet, der indeholder kort, er dels fordi det skal være muligt at tilføje kort til tavlen uden at være tilknyttet til et tegningslærred og dels på grund at det gør det lettere at tilknytte kortet til andre tegningslærreder. Desuden, har kort en reference til tegningslærredet, som består af et unikt id, som svarer til et tegningslærred. På denne måde kan et kort tilknyttes et tegningslærred, ved at referere til tegningslærredets id.

4.3.1 Tavle

Værktøjet skal kunne håndtere en whiteboardtavle med et variabelt antal tegninger og kort. Hver Kanban tavle består af et *Board* objekt som beskriver tavlen. *Board* klassen håndterer tavlens størrelse og titel, samt antal tegninger og

kort. *Board* klassen håndterer også tavlens position, men disse data gemmes ikke. Tavlens position ignoreres da tavlen skal være uafhængig af skærmopløsning og størrelse. Data håndteres og opdateres løbende vha. kontrolleren som i dette tilfælde er *Java Event Listeners* [16].

4.3.2 Tegningslærred

Canvas klassen repræsenterer et tegningslærred som kan indeholde flere forskellige tegningsformer. Tegningslærredet bruges til at rumme et sæt af tegningsformer, separeret fra tegningsformer som tilhører andre tegningslærreder. Tegningslærred virker også som et lag der er uafhængig af de andre lag. Dette er vigtigt da der ønskes at gruppere tegningsformer, således at de flyttes sammen som et enkelt objekt.

Tegningslærredet rummer tegningsformer i form af en afgrænsningsramme, som afgrænser tegningsformer via en *minimum bounding box* [19], som uddybes senere under afsnittet 4.8 Afgrænsningsramme. Det vil sige, at klassen tillader tegning og rummer alle de tegninger der tilhører sig. Dette indebærer også, at når tegningslærredet flyttes, så flyttes alle de forskellige tegningsformer der er inden for afgrænsningsrammen. Denne fremgangsmåde findes også i populære tegningsværktøjer, såsom photoshop.

Klassen indeholder tegninger af forskellige former, samt positionen og størrelsen på afgrænsningsrammen. Disse data opdateres løbende når der er fortaget en ændring.

4.3.3 Kort

Card klassen repræsenterer et kort, hvor dens position, størrelse og indehold kan håndteres og opdateres. Korts indhold består af tekst, som er blot en note, eller en brugerhistorie i en Kanban tavle. Kort kan også være en hel ny Kanban tavle som beskrives senere under afsnit 4.11 Kort hierarki. Kort objektet indeholder også et baggrundsfarve felt for at karakterisere kortets baggrund og type.

4.4 Koordinatsystem

Da systemet er delt efter MVC principper, er det en god ide at dele koordinat systemet i to lag, applikations og præsentations lag [20], hvilket gør at modellen er uafhængig af skærmopløsning. Modellen har sit eget fast koordinatsystem, mens præsentationen har sit eget variable koordinatsystem. Modellens koordinatsystem er 10000 x 7500, som er i 4:3 billedet format (4 enheder brede og 3 enheder høje). Modellens størrelse er baseret på, at der arbejdes med *integer* i stedet for *float*, da tavlens størrelse i brugergrænsefladen repræsenteres i *integers*. Der valgtes at bruge 10000 x 7500 i stedet for 1000 x 750, for at beholde placeringens og størrelsens præcision når tegninger og kort bliver skaleret ned til præsentationen [21]. I princippet kan man anvende større tal med 1:0,75 forhold, men i dette tilfælde antages at de valgte koordinatsystem størrelse dækker behovet.

Præsentationens koordinatsystem bestemmes ud fra den ønskede størrelse, som brugeren har valgt for tavlen (vinduet). Det betyder at når objekternes position og størrelse gemmes, så gemmes det i forhold til modellens koordinatsystem. Når modellen skal vises i præsentationen, så vises den i forhold til det koordinatsystem som præsentationen har.

Koordinater i præsentationen udledes fra de tilsvarende koordinater i modellen, og forholdet mellem modellen og præsentationen koordinatsystemers størrelse ud fra følgende formel:

$(x_{pres}, y_{pres}) = (x_{model} \cdot forhold_b, y_{model} \cdot forhold_h) = (x_{model} \cdot \frac{pres_{bredde}}{model_{bredde}}, y_{model} \cdot \frac{pres_{højde}}{model_{højde}})$

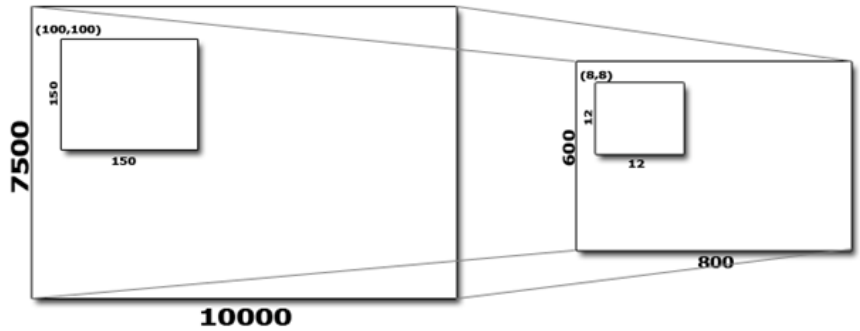
Dette vil virke både når præsentationen er større eller mindre end modellen.

Figur 4.3 på følgende side illustrerer hvordan tavlen samt dens tegningslærreder og kort bliver skaleret fra modellens koordinatsystem til præsentationen. Til venstre vises modellens koordinatsystem, og til højre vises præsentationslaget og hvordan modellen er er skaleret ned i forhold til dens størrelse.

Forholdet mellem modellen og præsentationen ifølge figuren beregnes som følgende:
$forhold_h = \frac{600}{7500} = 0,08$

$forhold_b = \frac{800}{10000} = 0,08$

Positionen (100,100) ganges med det tilsvarende forhold for at finde den tilsvarende position på præsentationen (8,8). Det samme gælder højden og bredden



Figur 4.3: Skalering af modellen til præsentationen

af tegningslærredet og kort.

Ændring af præsentationens størrelse sker ved at ændre vinduets størrelse, som vil automatisk resultere i ændring af forholdet mellem modellen og præsentationen. Dette vil også resultere i at skalerings størrelse vil ændre sig automatisk.

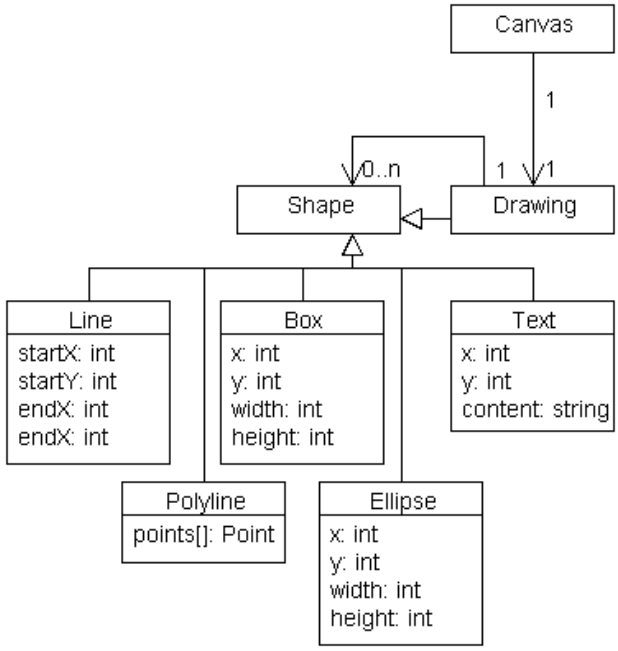
Skalering er baseret på denne fremgangsmåde, at når præsentations størrelse ændres, skaleres tegningslærred og kort automatisk.

4.5 Tegningsformer

På en Kanban tavlen har man friheden til at tegne frit, for at illustrere og forklare udviklingen. Derfor er det vigtigt at supplere værktøjet med flere forskellige tegningsformer, såsom linjer, polylinjer, firkanter, ellipse og tekst. Dette vil gøre tegning mere fleksibel og vil give brugeren flere tegnings muligheder.

Datastrukturen for tegningsformer er som illustreret i figur 4.4 på næste side. *Shape* klassen er en abstrakt klasse som beskriver en generel tegningsform. *Shape* klassen nedarves for at repræsentere en specifik tegningsform.

Canvas har en reference til *Drawing*, som har en liste der indeholder forskellige typer af tegningsformer. Enhver tegningsform har sin egen måde at blive tegnet på, alt efter hvilken form den repræsenterer. For eksempel, en linje har start og slut position, en firkant eller ellipse har bredde og længde osv.



Figur 4.4: Tegningsformers datastruktur

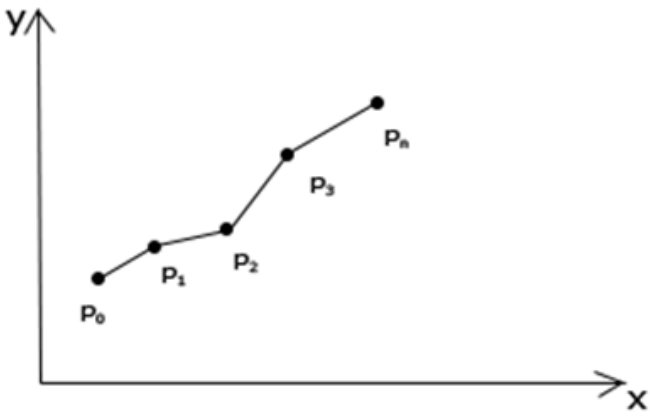
Hver tegningsform har sin egen datastruktur, som opdateres vha. af *getters* og *setters*. Baseret på hvilken form der skal tegnes, registreres inputs data og sendes videre til den valgte tegningsform objekt. Registrering af disse data vil blive uddybet under 4.6 Registrering af input data.

De næste afsnit vil beskrive hver tegningsform, hvad de repræsenterer og hvordan objektet bliver tegnet.

4.5.1 Polylinjer

Polyline klassen repræsenterer en fri-hånd-tegning, hvor formen bestemmes af musens bevægelse. Klassen gemmer musens positioner i form af punkter (*Point*) med de tilhørende koordinater.

Når objektet skal tegnes, så tegnes en linje mellem den nuværende og næste punkt. Figur 4.5 illustrerer hvordan polylinjerne tegnes ved at tegne en linje mellem det første og det andet punkt, og mellem det andet og det tredje punkt osv. igennem alle punkterne.



Figur 4.5: Tegne mellem polylinjerne

4.5.2 Linje

Line klassen repræsenterer en ret linje mellem to punkter. Klassen repræsenterer en mindre ugave af *Polyline* klassen, hvori *Line* gemmes kun en enkel linje med

start og slut punkt. Tegning af linjen er simplere end polylinjen, da der kun tegnes en enkel linje mellem to punkter.

4.5.3 Firkant

Box klassen repræsenterer en firkant, hvor formen bestemmes ud fra et start punkt samt bredden og højden. Bredden og højden bestemmes ved at beregne forskellen mellem start og slut punktet. Tegning af firkanten er naturligvis baseret på startpunktet samt bredden og højden.

4.5.4 Ellipse

Ellipse klassen repræsenterer en ellipse, hvor formen bestemmes på samme måde som firkanten. Klassen gemmer ellipsens startpunkt samt bredden og højden, som kan være forskellige fra hinanden.

4.5.5 Tekst

Text klassen repræsenterer en tekst. Klassen gemmer startpunktet samt teksten, og dermed tegning af teksten sker ved at tegne den ønskede tekst i den ønskede position.

4.6 Registrering af input data

Tegning af de forskellige tegningsformer forgår på brugergrænsefladen, og derfor er der brug for at registrere tegningsformens data for at gemme disse data i modellen.

Registrering af data forgår generelt således, at det starter når musens venstreklik holdes nede, fortsætter mens musen trækkes og slutter når musens venstreklik slippes.

Registrering af data er afhængig af hvilken tegningsform man valgte at tegne med. Er der tale om en polylinje så registreres alle musens positioner mens den trækkes rundt på tavlen. Er der tale om en linje, registreres kun start og

slut punkter som forgår henholdsvis når musen holdes og slippes. Firkanten og cirklens data registreres på samme måde, ved at gemme startpunktet når musens venstreklik holdes nede, mens højden og bredden beregnes i forhold til slutpunktet når musens venstreklik slippes. Tekstens data registreres ved at gemme startpunktet og ved at lytte på tastaturen, for at modtage og gemme den ønskede tekst.

Alle de ovennævnte data modtages i præsentationens koordinatsystem og derfor skaleres de op til modellens koordinatsystem før de gemmes i modellen. Skalering af præsentations data beregnes som beskrevet under afsnittet 4.4 Koordinatsystem .

4.7 Tegning og Gentegning

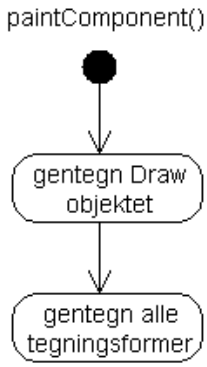
Repræsentationen af tavlens indhold, såsom tegninger og kort, er evigtig. Det derfor vigtigt at tegne indholdet på en billig og fleksibel måde, uden at kræve mange ressourcer og uden at miste billedets kvalitet. Da data bliver gemt i modellen, skal tegning og gentegning forgå direkte, ud fra modellen, på præsentationen. Dette kan gøres simpelt, da alle tegningsformer er gemt som data, såsom punkter, bredden højden osv., hvilket gøre at tegninger kan genkonstrueres ud fra modellen, når det er nødvendigt.

Når komponenten tegningslærredet, skal gentegnes, kalder Swing automatisk *paintComponent()* metoden. Denne metode kaldes, når vinduet får besked om at alle dens efterkommere skal tegnes, som beskrevet i den følgende artikel Painting in AWT and Swing [23].

Når *paintComponent()* metoden kaldes, gentegnes *draw* objektet, som indeholder alle de forskellige tegningsformer. Når *draw* objektet tegner tegningsformers listen, kan hver tegning gentegnes uafhængigt. Figur 4.6 på næste side viser at når metoden *paintComponent()* kaldes, normalt fra en *heavyweight* komponent (såsom *JFrame*), sendes beskeden videre for at gentegne *draw* objektet ud fra modellen.

4.8 Afgrænsningsramme

Kanban tavlen består af forskellige udviklingsfaser som repræsenteres i form af tegninger. Intentionen er at gruppere tegningsformer sammen ved at tilføje en



Figur 4.6: Gentegning af tegningsformer

afgrænsningsramme rundt om dem [19]. Afgrænsningsrammen er selve tegningslærredet som beskrevet før, hvor dens dimensioner beregnes ud fra de tilhørende tegningsformer.

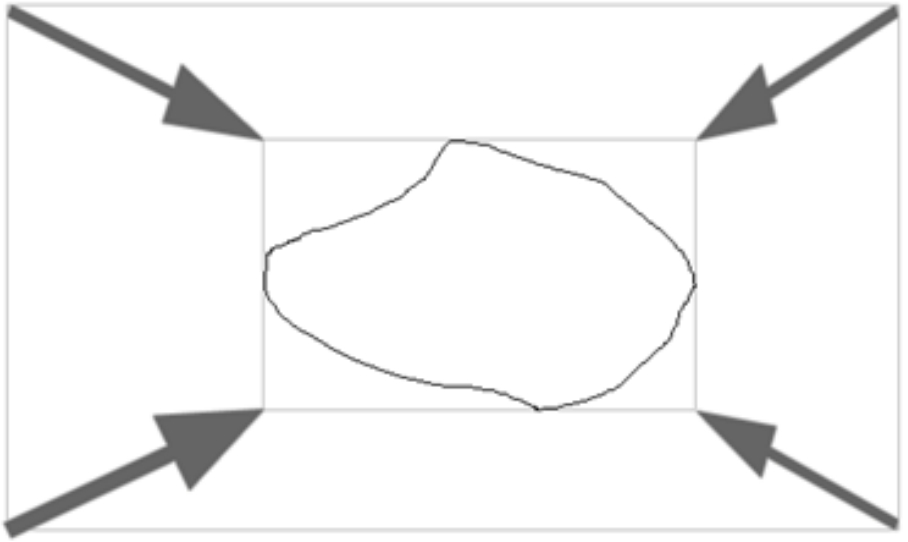
Når en ny tegningsform tilføjes til et tegningslærred, registreres de højeste og laveste x og y værdier. Ud fra disser værdier kan afgrænsningsrammens position, længde og bredde beregnes.

Når et nyt tegningslærred tilføjes, har afgrænsningsrammen tavlens størrelse, for at give brugeren muligheden for at tegne på hele tavlen. Når tegnetilstand afsluttes, ved at skifte til et andet tilstand, tilpasses afgrænsningsrammen i forhold til de tilhørende tegningsformer.

Figur 4.7 på følgende side illustrerer hvordan afgrænsningsrammens størrelse tilpasses. Størrelsen som kan ses på figuren er beregnet i forhold til polylinjen. Afgrænsningsrammen har dynamiske dimensioner, hvilket betyder at hvis man vælger at tegne en ny tegningsform i en allerede tilpasset afgrænsningsramme, så vil afgrænsningsrammen strække sig automatisk.

4.9 Gem og indlæs

Som beskrevet tidligere, er modellen implementeret som serialiserbar javabønner, der gør det nemmere at gemme og indlæse objekterne vha. henholdsvis serialisering og deserialisering af objekterne [24]. Serialisering bruges til at generere



Figur 4.7: Tilpasning af afgrænsningsrammens størrelse

en XML dokument som beskriver en serialiserbar javabønne objekt, mens deserialisering bruges til at generere objekter ud fra XML beskrivelsen. Det betyder at når Kanban tavlen gemmes, så gemmes kun modellens objekter i form af en XML fil.

Da *Board* objektet er det overordnet objekt, tager serialisering udgangspunkt i dette objektet og gemmer alle dens egenskaber (bl.a. tegningslærreder og kort). Tegningsformer gemmes som data i stedet for billeder, hvilket betyder at hver tegningsform gemmes som forklaret i de forrige afsnit. For eksempel, polylinje gemmes som punkter, linjen gemmes som to punkter, teksten gemmes som punkt og en streng osv.

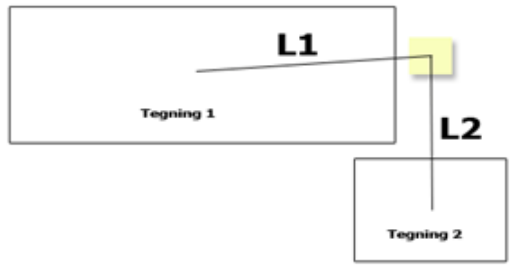
Når *Board* objektet loades, genereres modellen og derefter genkonstruerer systemet præsentationen automatisk ud fra modellen.

4.10 Kort tilknytning

I Kanban, er det essentielt at kort kan tilknyttes tegninger, på en måde der indikerer hvilket kort tilhører hvilken tegning. Derfor er det oplagt at have denne funktionalitet med, men målet er også at gøre tilknytning nem og automatisk.

Værktøjet skal have en form af intelligens, således at værktøjet selv tilknytter kort når det flyttes. Det betyder at der skal være en specifik regel for at bestemme hvilket kort der skal tilknyttes til hvilket tegningslærred.

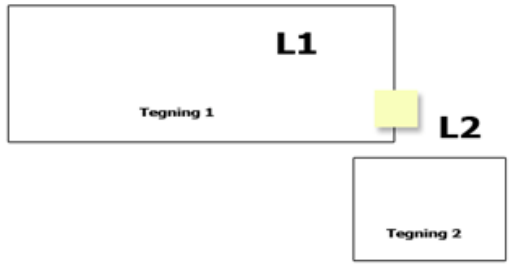
En måde at gøre på, er at tilknytte kortet til den nærmeste tegning. Den nærmeste tegning kan beregnes vha. afstandsformlen mellem korts og tegnings centrum. Denne måde virker fint i de fleste tilfælde, men den er ikke hel fejlfri, da i nogle tilfælde tilknyttes kortet ikke til den ønskede tegning. Det er i tilfælde hvor afstanden til den ønskede tegnings centrum er større end andre. Figur 4.8 illustrerer det tilfælde hvor afstanden til tegning 1, som er det ønskede tegningslærred til at knytte med, er større en afstanden til en tegning 2. Dette resulterer i, at kortet bliver tilknyttet til den forkerte tegning. Der er også en anden problematik ved denne måde, da kort ikke kan tilføjes på tavlen uden at blive tilknyttet til den nærmeste tegningslærred, hvilket gør at denne måde er ikke brugbar.



Figur 4.8: Kort tilknyttes det forkerte tegningslærred

En anden måde at tilknytte kort på er, når kortet skærer igennem en afgrænsningsramme. Denne måde sikrer at kortet tilknyttes til den ønskede tegning. Beregning af skæring beregnes i forhold til afgrænsningsramme, hvor beskæring beregnes ved at tjekke om kortet ligger inden for afgrænsningsrammen.

Figuren 4.9 på følgende side viser, at kortet er tilknyttet tegningslærredet 1 fordi det skærer igennem afgrænsningsrammen, selvom afstanden til tegningslærredet 2 er kortere. Den sidstnævnte måde vælges, da det er sikrere at tilknytte med, og det er muligt at have kort på tavlen som ikke er tilknyttet til et tegningslærred.



Figur 4.9: Kort tilknyttes det ønskede tegningslærred

4.11 Kort hierarki

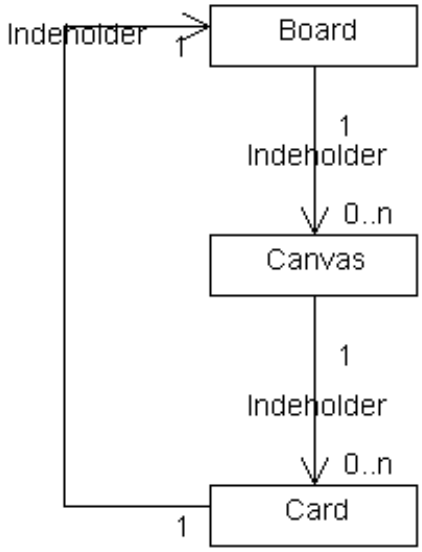
En af de ting som blev identificeret som mangel i de andre Kanban værktøjer og som en vigtig udvidelse i dette speciale, er at et kort kan også repræsentere en hel nye Kanban tavle. Denne udvidelse vil give muligheden at have en Kanban tavle på højniveau, hvor uddybende detaljer kan ses ud fra korts indhold.

Figur 4.10 på næste side illustrerer hvordan systemet er ændret til at understøtte denne udvidelse. Det opnås ved at *Card* objektet har reference til et nyt *Board* objekt, som repræsenterer modellen for en ny tavle. Der er ingen begrænsning for dybden af hierarkiet og derfor er det muligt for kort at indeholde en tavle, som indeholder et kort der også indeholder en tavle osv.

Den nye tavle bliver gemt ind i kort objektet og vil først blive initialiseret når der vælges at udvide kortet som en nye tavle.

4.12 Sletning

Sletning af tegningslærred svarer til at viske en del af tavlen ud, og sletning af kort svarer til fjernelsen af kort. Da interaktionen forgår i brugergrænsefladen, registreres først hvilken komponent der ønskes slettet og derefter slettes det tilsvarende objekt i modellen. Slette mekanismen sørger for at modellen og præsentationen er synkroniseret når en komponent slettes.



Figur 4.10: Kort hierarki

4.13 Tekst redigering

Som beskrevet før, understøtter værktøjet at tilføje tekst til tegninger. Denne funktionalitet er udvidet med en tekst redigeringstilstand, som tillader brugeren at redigere en allerede eksisterende tekst. Først registreres hvilke tekst der skal redigeres, ved at beregne afstanden mellem musens venstreklik og den tætteste tekst. Derefter modtages tastaturens input og sendes videre til det valgte tekst objekt.

4.14 Fortryd & gentag

Det er vigtigt at værktøjet tillader at fortryde eller gentage en handling, som kan ske ved en fejl eller misforståelse. Fortrydelse og gentagelse af en handling vil hjælpe brugeren med at rette fejl og med at prøve forskellige aspekter af værktøjet.

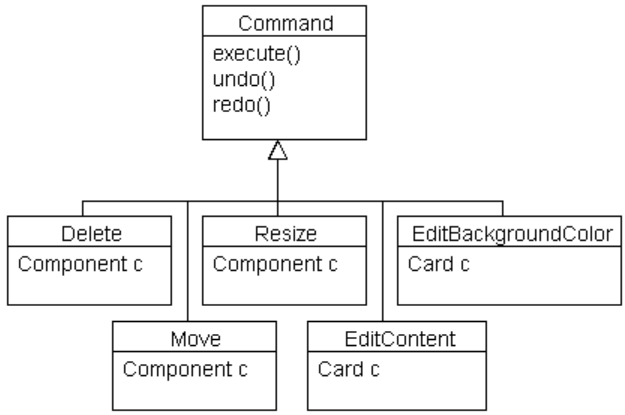
For at opnå denne funktionalitet, skal aktioner betragtes som atomare handlinger, som skal gemmes for at kunne fortryde eller gentage dem senere. Fortrydelses og gentagelses mekanismen er baseret på *command pattern* [25], som er et designmønster, hvor en metode repræsenteres af et objekt. Command pattern består generelt af 3 dele, en klient som instantierer *command* objektet, en tilkalder som bestemmer hvornår metoden skal kaldes og en modtager som er instansen af command objekt klassen, der indeholder metodens kode.

Intentionen er at opnå en multi-niveau fortryde og gentage, hvilket betyder at specifikke handlinger bliver implementeret som *command* objekter, og værktøjet sørger for at gemme hvert *command* objekt i en stak, som kan geneksekveres senere. Figur 4.11 på følgende side viser hvordan *command* objekter er struktureret og hvilke handlinger der understøttes.

Det er muligt at fortryde og gentage sletning af et komponent (tegningslærred eller kort). Det samme gælder flytning og størrelsestilpasning. Tilføj handlingen er undtaget, da værktøjet allerede har en slette funktion.

Korts indhold og baggrundsfarve kan ændres og hermed fortrydes eller gentages, ved at gemme teksten og farven før og efter ændring.

Mekanismen er baseret på at gemme kun de nødvendige informationer, for eksempel at gemme komponentens position før og efter flytning, og derefter kan den tidligere position genskabes. Det er en billig måde at gemme en handling



Figur 4.11: Command datastruktur

på, sammenlignet med at gemme hele *Board* objektet.

Fortrydelse og gentagelse påvirker modellen direkte og derefter opdateres præsentationen, således at præsentationen afspejler modellen når en handling fortrydes eller gentages.

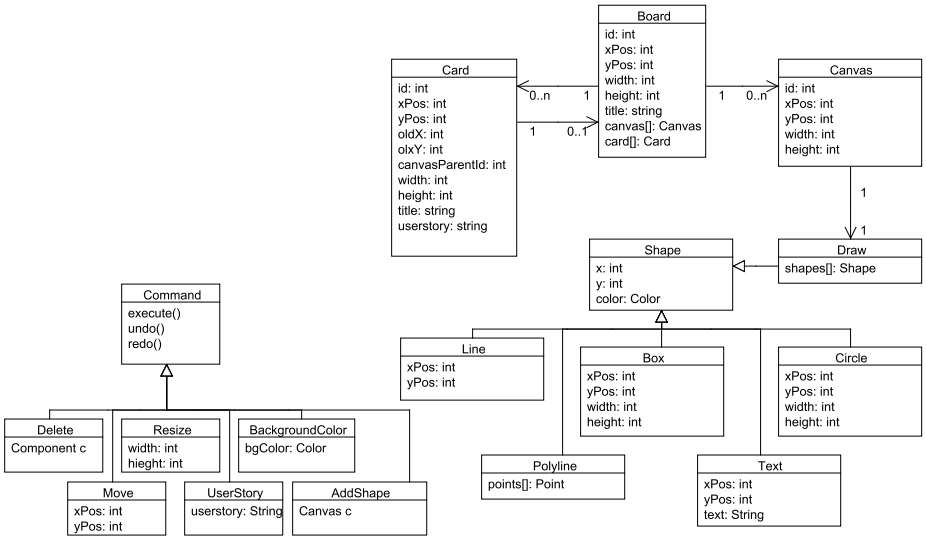
Derudover var det muligt at fortryde og gentage tegningsformers handlinger. Men da denne funktionalitet blev tilføjet først, blev den implementeret ifølge en billigere løsning end *command pattern*, nemlig *memento pattern* [26]. *Memento* er et software designmønster som bruges til at genskabe et objekt fra en tidligere tilstand. Systemet indeholder to objekter, en *originator* og en *caretaker*. En *originator* er dette objekt som bruges af systemet, mens *caretaker* er den som bruges til at gemme kopier af *originator*'ens tilstand hen ad vejen. Når *originator* objektet ændres, vil den tjekke ud og dermed gemmes en kopi af den i *caretaker*. Men at have to forskellige fortrydelses mekanismer er ikke en pæn design, og det er bedst hvis *memento* erstattes med *commaned*. Derfor blev *memento* erstattet med *command*, og fortrydelse samt gentagelse af tegningsformer betragtes nu som atomar handling som gemmes i stakken.

At implementere to forskellige fortrydelses mekanismer, samt at erstatte *memento* med *command*, har krævet ekstra arbejde. Det skyldtes, at den første brugerhistorie som blev implementeret, handlede om at implementere fortrydelse og gentagelse specifikt til tegningsformer. Den simpleste løsning var, at anvende memento. Senere i forløbet valgtes at udvide denne funktionalitet således at det også understøttes de førnævnte handlinger. Baseret på det valg, er *command* nu

den eneste fortrydelses mekanisme, som bruges i værktøjet.

4.15 Klassefordeling

Figur 4.12 viser hvilke klasser som systemet består af. Der er 5 hovedklasser, *Board*, *Canvas*, *Card*, *Shape* og *Command*. Værktøjet indeholder en liste af *command*'s objekter, således at handling fortrydes i forhold til tavlen. Enhver tavle har en stak af handlinger, og hvis et kort udvides som en ny tavle, har den nye tavle også sin egen stak.



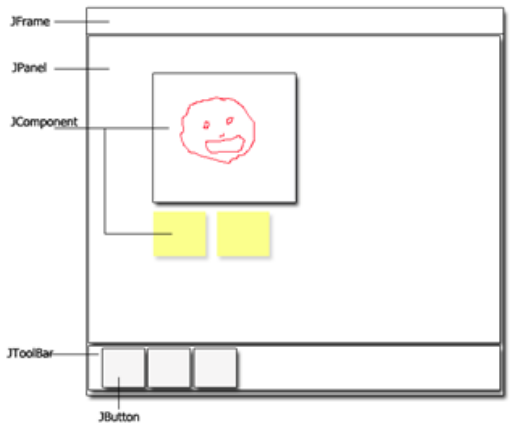
Figur 4.12: Klassediagram

4.16 GUI opdeling

Der er givet nogle tanker og overvejelser til hvordan den grafiske brugerflade skulle opdeles og hvilke komponenter skulle bruges til hver del. Ligesom det er vigtigt at lave et pænt design, er det også vigtigt at designe hvordan den grafiske brugerflade skal struktureres.

Figur 4.13 på følgende side viser en skitse over hvordan whiteboardtavlen kan opdeles. Der vises også hvilken Swing komponent der tænkes at repræsentere

hvilken del.



Figur 4.13: GUI opdeling

JFrame og *JPanel* delen bruges til at repræsentere tavlen, hvor *JFrame* er selve vinduet og *JPanel* er tegneområdet. Vinduet afspejler tavlens titel, position og størrelse mens *JPanel* indeholder tegningslærreder og kort.

Tegninger og kort er et skræddersyet *JComponent*. Nederst i figuren ses en værktøjslinje som indeholder knapper, hvor hver udfører en speciel handling. Værktøjslinjen er en *JToolBar* og knapperne er *JButton*.

4.17 Opsummering

Dette kapitel beskriver de overordnet designvalg for at løse de problemer og udfordringer som blev identificeret i analysen. Der beskrives hvordan systemet er bygget op, og hvordan komponenterne er strutkureret i forhold til hinanden. Der er også givet en forklaring til de forskellige designmønstre, samt hvordan de udnyttes til at designe dette værktøj. Der er vist hvordan whiteboardtavlen, tegningslærreder og kort er designet til at tjene deres mål. Der vises også et klassediagram der viser hvilke klasser værktøjet består af.

Kapitel 5

Implementering

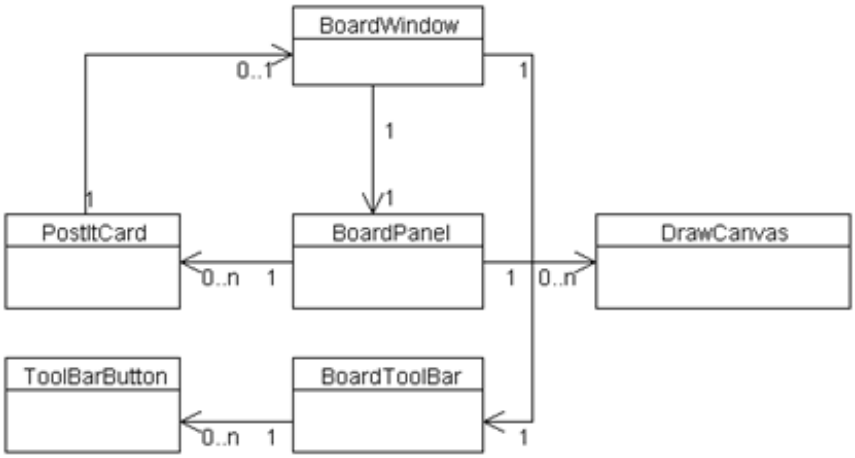
I dette kapitel vil der være en detaljeret gennemgang af de vigtigste dele implementeret i værktøjet. Der vil blive introduceret de vigtigste implementeringsprincipper og koncepter der er anvendt i dette speciale. Værktøjet er implementeret i Java, og alle kode eksempler som vises i dette kapitel er Java kode.

5.1 Opbygning

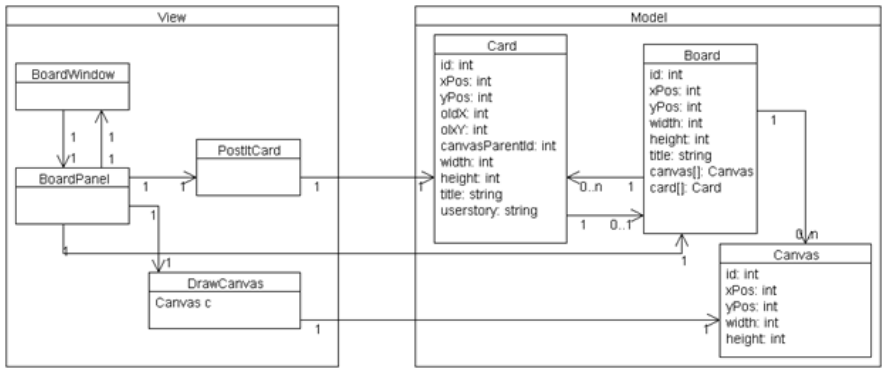
Som beskrevet i design, er systemet delt efter MVC principper i to dele, modellen og præsentationen, der er forbundet gennem kontrolleren. Figur 4.12 på side 42 under Klassefordeling viste hvordan modellen er bygget. Figuren kan relateres til Figur 5.1 på følgende side, som repræsenterer præsentationen.

Figur 5.1 på næste side viser præsentations struktur, hvor *BoardWindow* er hoved objektet og sammen med *BoardPanel* repræsenter de tavlen. *DrawCanvas* repræsenterer et tegningslærred, mens *PostItCard* repræsenterer et kort.

Figur 5.2 på følgende side viser hvordan systemet er delt efter MVC, og hvordan præsentationen interagerer med modellen og hvilke klasser i præsentationen der har reference til modellen. *BoardWindow* instantierer alle systemets dele når værktøjer startes og sørger for alle objekterne er oprettet og forbundet.

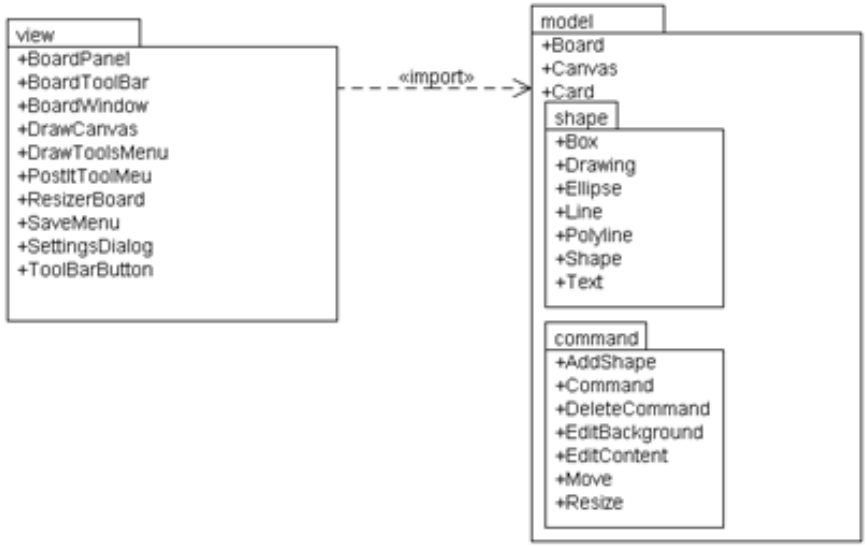


Figur 5.1: Præsentations opbygning



Figur 5.2: Interaktion mellem view og model

Derudover består koden af to hoved pakker, *model* og *view*. Modellen indeholder to ekstra underpakker, *shape* og *command*, som henholdsvis repræsenterer tegningsform og *command* objekter. Figur 5.3 viser pakkernes struktur.



Figur 5.3: Systemets pakker

De næste afsnit vil beskrive hver del af systemet og hvordan de er implementeret.

5.2 Tavle

Kanban tavlen er implementeret således, at den består af to dele, et vindue og et tegneområde. Vinduet repræsenterer tavlens position og størrelse, mens tegneområdet indeholder tegningslærreder og kort. Figur 5.4 på følgende side viser opdeling af tavlen med en ydre ramme der repræsenterer selve vinduet og en indre ramme der indeholder tegninger og kort.

Figur 5.4: Tavle

Vinduet henter sin tilstand fra *Board* objektet, såsom titel, position og størrelse. Koden 5.1 viser et eksempel på hvordan tavlens titel hentes fra modellen:

```
1 this.setTitle(board.getTitle());
```

Listing 5.1: Tavlens tilstand hentes fra modellen

Dette gælder også når titlen ændres, så ændres modellen først og derefter henter tavlen den nye titel fra modellen. Denne type af ændringer sker direkte på modellen, da præsentationen har en direkte reference til den førstnævnte.

Vinduet implementerer *ComponentListener* som lytter på tavlens position og størrelse. Når disse ændres, opdateres de tilsvarende egenskaber i modellen automatisk. *ComponentListener* er selve kontrolleren som observerer ændringer i brugergrænsefladen og opdaterer board objektet i modellen.

Koden 5.2 viser hvordan kontrolleren opdaterer board objektet når tavlens størrelse ændres.

```
1 @Override
2 public void componentResized(ComponentEvent e) {
3         board.setAppWidth(e.getWidth());
4         board.setAppHeight(e.getHeight());
5 }
```

Listing 5.2: Kontrolleren opdater modellen

Denne størrelse bruges til at beregne hvilken størrelse præsentationen har, og dermed beregne forholdet mellem den og modellen, for at skalere objekter fra modellens til præsentationens koordinatsystem.

Vinduet bruger en standard *BorderLayout* til at bestemme placering af tegneområdet øverst, og værktøjslinjen nederst. Derimod har tegningsområdet *BoardPanel* ingen *BorderLayout* og det er fordi det skal være muligt at placere tegningslærred og kort et vilkårligt sted på tegneområdet.

Tegneområdet *BoardPanel* indeholder en *ArrayList* af den grafiske repræsentation af tegningslærreder og kort. Det vil sige, at de grafiske objekter af tegningslærred og kort bliver gemt i *BoardPanel (view)*, mens data bliver gemt i *Board* objektet (model). Tegningsområdet sørger for at synkronisere præsentationens og modellens objekter.

5.3 Værktøjets tilstande

Værktøjet har forskellige tilstande som hver bestemmer hvordan brugerinteraktionen behandles af systemet. Systemet lytter på musens interaktion for at bestemme om en komponent skal flyttes, størrelsestilpasses eller tegnes på.

Der er brugt følgende måder til at bestemme hvordan musens interaktion skal fortolkes og behandles, for at udføre den ønskede handling. Der er tre forskellige tilstande, som kan aktiveres via værktøjslinjen, beskrives som følgende:

1. **Tegningstilstand**: Denne tilstand aktiveres når brugeren ønsker at tegne på et tegningslærred. Tilstanden vil gøre at værktøjet behandler musens interaktion og fortolker den til tegningsform.

2. **Flyttetilstand**: Denne tilstand aktiveres når brugeren ønsker at flytte et tegningslærredet eller et kort.

3. **Størrelsestilpasningstilstand**: denne tilstand aktiveres når brugeren ønsker at tilpasse størrelsen af kort.

Aktivering af den ene tilstand vil automatisk resultere i inaktivering af de andre tilstande, som den nedenstående kode 5.3 viser, således at der er altid én og kun én tilstand som er aktiv.

```
1  public void allowDrawing(boolean drawing){
2          allowDrawing = drawing;
3          allowMoving = false;
4          allowResizing = false;
5  }
6  public void allowMoving(boolean moving){
7          allowMoving = moving;
8          allowDrawing = false;
9          allowResizing = false;
10 }
11 public void allowResizing(boolean resizing){
12         allowResizing = resizing;
13         allowMoving = false;
14         allowDrawing = false;
15 }
```

Listing 5.3: Aktivering af systemets tilstande

5.4 Tegningslærred

Tegningslærredet er et skræddersyet *JComponent* og grundet til det er, at der bruges mange at de funktioner og *listeners* som *JComponent* allerede har og derfor slippes for at genimplementere dem.

Tegningslærred implementerer en *MouseMotionListener* [27], som bruges til at lytte på musens bevægelse og dermed bestemme hvor komponenten skal flyttes til.

```
1 selectedComponent = e.getSource();
2 Rectangle r = selectedComponent.getBounds();
3 r.x += e.getX() - offset.x;
4 r.y += e.getY() - offset.y;
5 this.setBounds(r);
```

Listing 5.4: Flytning af tegningslærred

Koden i 5.4 viser hvordan komponenten lytter til musen og bestemme hvilken komponent som ønskes at flytte, hvorefter opdaterer komponentens position opdateres i forhold til musens bevægelse.

På samme måde som vinduet, bruger tegningslærred *DrawCanvas* en *ComponentListner*, for at lytte på komponentens position og størrelse, for at opdatere modellen når der fortages en ændring:

```
1 @Override
2 public void componentMoved(ComponentEvent e) {
3         drawCanvas.setAppLayerXPos(e.getComponent().getX());
4         drawCanvas.setAppLayerYPos(e.getComponent().getY());
5 }
```

Listing 5.5: Kontrolleren opdaterer tegningslærredets model

Når positionen gemmes i modellen, så gemmes den i modellens koordinatsystem, som kan ses i kode 5.6:

```
1 public void setAppLayerXPos(int xPos){
2         xPos = (int)(xPos/(board.getWidthScale()));
3 }
4 public void setAppLayerYPos(int yPos){
5         yPos = (int) (yPos/(board.getHeightScale()));
6 }
```

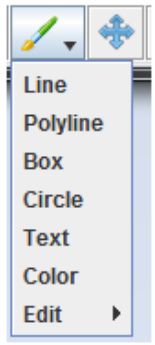
Listing 5.6: Skalering af præsentations til modellens koordinater

Komponenten implementerer også en *MouseListener*, som primært benyttes til at lytte på musen, når der tegnes. Komponenten lytter mens musen trækkes på, for at fremstille den ønskede tegnings form. Denne funktionalitet uddybes yderligere i det næste afsnit.

5.5 Tegningsformer

Brugeren kan igennem grænsebrugerfladen bestemme hvilken tegningsform der ønskes at tegnes med. Aktivering af en tegningsform, medføre at fortolkning af musens bevægelse er i overensstemmelse med formen, som beskrevet under 4.6 Registrering af input data.

Tegningslærredet vil sørge for at vise tegningsformen mens den bliver tegnet, således at brugeren får en fornemmelse af, hvordan tegning forgår, og hvordan den endelige form er. For eksempel, når man vælger at tegne en linje, vil linjens endepunkt følge med musen, og brugeren vil være i stand at se linjens form, mens musen bevæger sig. Denne mekanismen virker ved at opdatere modellens data og bede om at gentegne objektet. Den endelige form af tegningsformen bestemmes først når musens venstreklik slippes.



Figur 5.5: Tegningsform valgmuligheder

Tegningslærredet udnytter de allerede eksisterende tegne metoder som Graphics objektet har. Graphics objektet modtages som parameter i *paintComponent()* metoden og bruges til at tegne hver tegningsform. Følgende kode viser hvordan hver tegningsform bliver tegnet ved hjælp at Graphics objektet.

At tegne en linje:

```
1 graphics2D = (Graphics2D) g;
```

```
2 graphics2D.drawLine(x0, y0, x1, y1);
```

Listing 5.7: Tegne en linje

At tegne en firkant:

```
1 graphics2D.drawRect(x, y, width, height);
```

Listing 5.8: Tegne en firkant

At tegne en ellipse:

```
1 graphics2D.drawArc(x, y, width, height,0, 360);
```

Listing 5.9: Tegne en ellipse

At tegne en tekst

```
1 graphics2D.drawString(text, x, y);
```

Listing 5.10: Tegne en tekst

Det er også muligt at bestemme hvilken farve en tegningsformen skal tegnes med, ved at vælge farven inden tegningen påbegyndes. Shape klassen har et farve felt, som kan opdateres for at afspejle den valgte farve. Farven sættes inden draw metoderne kaldes:

```
1 graphics2D.setPaint(drawColor);
```

Listing 5.11: Sætter tegningsfarven

5.6 Kort

Kort komponenten er et skræddersyet *JComponent*, af samme årsag som tegningslærredet er det. Kort komponenten implementerer *MouseListener* som tegningslærred, for at kontrollere hvor kortet skal flyttes til på tavlen.

Komponentens dimensioner kan ændres ved at trække i kanterne. Som brugervenligheds hjælp, ændres musens ikon når musen nærmere sig kanterne, for at indikere at det er muligt at trække i kanterne. Dette gøres ved at lytte på musens position og når musen nærmer sig en kant (højre, venstre, top eller bund),

vil der vises en tilsvarende museikon, der indikerer hvilken side der kan hæves i. Følgende kode 5.12 viser at når musen nærmer sig en kant, ændres cursorens ikon:

```
1  if(p.y  < 5) {
2  setCursor(Cursor.getPredefinedCursor(Cursor.N_RESIZE_CURSOR));
3  }
4  else if (p.x < 5){
5  setCursor(Cursor.getPredefinedCursor(Cursor.W_RESIZE_CURSOR));
6  }
7  else if (p.x > this.getWidth()-5){
8  setCursor(Cursor.getPredefinedCursor(Cursor.E_RESIZE_CURSOR));
9  }
10 else if (p.y > this.getHeight()-5){
11 setCursor(Cursor.getPredefinedCursor(Cursor.S_RESIZE_CURSOR));
12 }
13 else if (allowMoving){
14 setCursor(Cursor.getPredefinedCursor(Cursor.MOVE_CURSOR));
15 }
16 else {
17 setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
18 }
```

Listing 5.12: Ændrer musens ikon når musen nærmer sig en kant

Baseret på hvilken museikon der er sat, kan komponenten bestemme hvordan kortets dimensioner skal ændres. Det indebærer at når musens ikon for eksempel har *N_RESIZE_CURSOR*, er det toppen der skal ændres. Derefter lyttes på musens bevægelse for at beregne kortets størrelse. Kode 5.13 viser hvordan kortets størrelse ændres når musen trække ved en kant.

```
1  int type = getCursor().getType();
2  Point p = offset;
3  int dy = e.getY()-p.y;
4  int dx = e.getX()-p.x;
5
6  switch(type) {
7  case Cursor.N_RESIZE_CURSOR:
8          setBounds(getX(), getY()+dy, getWidth(), getHeight()-dy);
9          break;
10 case Cursor.S_RESIZE_CURSOR:
11         setBounds(getX(), getY(), getWidth(), getHeight()+dy);
12         offset = e.getPoint();
13         break;
14 case Cursor.W_RESIZE_CURSOR:
15         setBounds(getX()+dx, getY(), getWidth()-dx, getHeight());
16         break;
17 case Cursor.E_RESIZE_CURSOR:
18         setBounds(getX(), getY(), getWidth()+dx, getHeight());
19         offset = e.getPoint();
20         break;
21
```

```
22 default:
23         System.out.println("unexpected type: " + type);
24 }
```

Listing 5.13: Ændring af korts størrelse

Korts indhold kan indsættes ved at dobbeltklikke på kortet, hvorefter åbnes redigeringsvinduet, som kan ses i figur 5.6. Teksteditoren er en simpel *JEditorPane*, som er modtagelig for multilinjer tekst.

Figur 5.6: Korts indhold og visning

Når teksten indsættes i teksteditoren, gemmes den efterfølgende i modellen og vises på kortet. Teksten på kortet, som kan ses på figur 5.6, brydes i forhold til kortets størrelse, således at teksten kan læses på kortet.

Tekst bredde beregnes vha. *FontMetrics* og derefter sammenlignes med kortets bredde, for at bestemme hvor teksten brydes. Teksten brydes kun når der er et mellemrum mellem ordene.

5.7 Skalering

Skalering af tavlen og dens tilhørende komponenter er afhængig af vinduets størrelse. Ved hjælp af *ComponentListener* lyttes på vinduets størrelse. Når størrelsen ændrer sig, ændres forholdet mellem modellens og præsentationens koordinatsystem, hvorefter gentegnes tegninger og kort.

Det er et kendt problem, at et billede vil miste en del af detaljerne, når det skaleres. Dette problem løses ved at gentegne hele billedet i de nye dimensioner, som komponenten får efter skaleringen. Sagt med andre ord, tegningsformer bliver gentegnet i forhold til den skalerede størrelse, for at beholde alle tegningsformens detaljer.

Når vinduet skaleres, kaldes metoden *componentResized* automatisk. Metoden sørger for at opdatere præsentationens koordinatsystem størrelse, og derefter beder den om at gentegne tegninger og kort, som ses i kode 5.14

```
1 @Override
2 public void componentResized(ComponentEvent arg0) {
3         //opdaterer view'ets koordinatsystem
4         board.setAppWidth(getWidth());
5         board.setAppHeight(getHeight());
6         //gentegn tegninger og kort
7         panel.resizeCanvas();
8         panel.resizeCards();
9 }
```

Listing 5.14: Skalering af tegningslærreder og kort

Skalering af positioner og tegninger forgår ved at gange de oprindelige værdier (x, y, højde, bredde) med forholdet mellem modellens og præsentationens koordinatsystem. Skalering forgår omgående og der er ingen begrænsninger for den minimale eller maximale skalerings størrelse.

5.8 Gem og indlæse

For at kunne gemme tavlens tilstand, er det nødvendigt at serialisere tavlen samt tegningslærreder og kort til et kendt format. Tavlens tilstand kan generelt gemmes, ved at have en metode der kan generere en lang streng baseret på et *Board* objekt (serialisering), og en metode der genererer et *Board* objekt ud fra en lang streng (deserialisering). Dette kan gøres på 3 forskellige måder:

1. **Implementering af serialisering og deserialisering metoderne**:

   Her skal metoderne for serialisering og deserialisering af objekterne skrives fra bunden af. Fordelen ved denne løsning er, at der er kontrol over filformatet, og hvordan tavlen gemmes, hvilket sikrer en fremadkompatibelt med nyere udgaver af værktøjet på alle Java platforme. Ulempen er at det kræver en del at designe et filformat og at man selv skal implementere serialiserings og deserialiserings metoderne.

2. **At benytte Javas serialiserings API**:

   Her benyttes et Java API til at specificere, hvad der skal gemmes, og Java vil håndtere hvordan det gemmes. Fordelen er, at det er let at serialisere og deserialisere metoderne, da Java selv bestemmer filernes format. Ulempen er, at filformatet ikke kan garanteres at være fremadkompatibelt, hverken med nyere udgaver af Java eller nyere udgaver af værktøjet. Brud med kompatibiliteten kan forekomme når klassens implementering ændres ved at slette et felt, ændre ikke-statiske felt til statisk eller ændre ikke-transient felt til transient osv.

3. **At benytte XMLEncoder**:

   Her benyttes *XMLEncoder* til at gemme javabønner. Javabønnernes egenskaber (som tilgås via get og set metoder) bruger *XMLEncoderen* til at serialisere objekter. Fordelen ved denne løsning er at det er relativt nemt at skrive serialiserings og deserialiserings metoderne og at filformatet er fremadkompatibelt. Ulempen er at alle objekter i modellen, på nær arrays, skal være javabønner, og at XML filen typisk fylder mere end en traditionelle binær løsning.

Der blev valgt at benytte *XMLEncoder*, da på den måde slippes for alt besværet med at designe filformatet og implementering af serialisering og deserialisering, og at filformatet vil være fremadkompatibelt. Valget skyldes også at der allerede valgtes at designe modellens objekter som javabønner.

5.8.1 Sammenlignings metode

For at hjælpe brugeren, advarer værktøjet hvis det lukkes ned, uden at gemme de sidste ændringer. Når brugeren vælger at lukke værktøjet, modtager brugeren en meddelelse om at gemme tavlen, hvis tavlen er ikke gemt eller hvis tavlen er ændret i forhold til den sidste gemte fil.

Det er relativt nemt at advare brugeren, i tilfældet at brugeren ikke har gemt tavlen; værktøjet vil sørge for at undersøge om der er oprettet nye objekter (tegningslærreder eller kort).

Tabel 5.1: Samenlignings variabler

Tegninger	
Antal tegningslærreder	Hvor mange tegningslærreder der er oprettet
Id	Ethvert tegningslærreds id
Position	X og y værdier for ethvert tegningslærred
Størrelse	Længden og bredden for ethvert tegningslærred
Antal tegningsformer	Hvor mange tegningsformer ethvert tegningslærred indeholder
Kort	
Antal kort	Hvor mange kort der er oprettet
Id	Ethvert korts id
Position	X og y værdier for ethvert kort
Størrelse	Længden og bredden for ethvert kort
Indhold	Indholdet af ethvert kort
Tilknytning	Hvilket tegnings ethvert kort er tilknyttet med
Baggrundsfarve	Kortets baggrundsfarven
Tavle-hierarki	Hvis kort er udvidet som en tavle

Sagen er sværere hvis tavlen er allerede gemt, og der fortages nye ændringer. Værktøjet skal kunne sammenligne den nuværende tavle med den gemte. Dette kræver at skrive en ny sammenlignings metode fra bunden af, for at specificere hvordan de to objekter kan sammenlignes, og hvornår de to objekter anses som forskellige.

Tabel 5.1 viser hvilke værdier der sammenlignes. Værktøjet sammenligner de ovenstående variabler mellem tavlen og den gemte fil. Hvis der er en enkel variable der ikke i overensstemmelse i de to objekter, betragtes det som ændring og derfor advares brugeren om at gemme. Sammenligningsmetoden får det gemte *Board* objekt som parameter, og dermed sammenlignes de to objekter. I tilfældet hvor kort er udvidet som tavle, kaldes sammenlignings rekursivt, for at sammenligne mellem de to *Board* objekter.

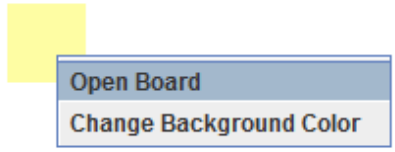
5.9 Kort hierarki

Kort komponenten har et ikke-initialiseret *Board* objekt, som først initialiseres når der vælges at udvide kortet som en tavle. Ved at højreklikke på kortet, fås en mulighed at åbne den nye tavle, som figur 5.7 på følgende side. Hvis *Board*

objektet ikke er initialiseret, vil den blive initialiseret, ellers åbnes en ny kanban tavle med det *Board* objekt som kortet allerede har, som følgende kode 5.15 viser:

```
1  public void startNewBoard(Card card){
2          if(card.getCardBoard() == null){
3                  BoardWindow boardWindow = new BoardWindow();
4                  card.setCardBoard(boardWindow.getBoard());
5          }
6          else {
7                  BoardWindow boardWindow = new BoardWindow();
8                  boardWindow.setBoard(card.getCardBoard());
9          }
10 }
```

Listing 5.15: Oprettelse og visning af korts tavlen



Figur 5.7: Udvide kort som tavle

5.10 Fortrydelse og gentagelse

For at kunne gå tilbage til en tidligere tilstand, er det nødvendigt at gemme tidligere tilstande. Det er åbenlyst at når en handling fortrydes, forventes at tavlen gendannes til den forrige tilstand og derfor er det oplagt at gemme disse handlinger i en stak. Her benyttes en fortryd stak som beskriver tidligere tilstande, og en gentage stak som beskriver tilstande man har fortrudt. Når der udføres en handling, pushes et objekt på fortryd stakken. Hvis brugeren fortryder, kan de tidligere tilstande genereres fra objekter på fortryd stakken. Mens der vendes tilbage til den tidligere tilstand poppes objekterne fra fortryd stakken og pushes på gentag stakken. Hvis brugeren så fortryder at have fortrudt en handling, kan brugeren igen gå tilbage med objekterne på gentag stakken.

Objekterne der indsættes på stakken skal beskrive de tidligere tilstande. Dette kan gøres absolut, ved at gemme hele tavlens tilstand, eller relativt, ved at gemme ændringerne i tavlens tilstand. Der blev valgt, at gemme tavlens tilstand

relativt, da en enkelt handling fra en bruger typisk ikke vil ændre hele tavlens tilstand, og man derfor kan beskrive tidligere tilstande med meget mindre hukommelse. At beskrive tilstandene relativt svarer I princippet til at benytte deltakomprimering på tilstandene i den rækkefølge de oprindeligt blev udført. Dette betyder at ændringer af tavlens tilstand beskrives vha. *command* objekter, og at den oprindelige tilstand (start tilstand) kan gendannes ved at fortryde alle *command* handlinger. Det indebærer at *command* objektet udfører en atomar handling der svarer til en lille ændring af tavlens tilstand.

Følgende handlinger i tavlen betragtes som atomare handlinger:

1. Flytte tegningslærred eller kort
2. Slette tegningslærred eller kort
3. Tilpasse størrelsen af kort
4. Ændre korts indhold
5. Ændre korts baggrundsfarve
6. Tilføje tegningsform

Disse handlinger er implementeret som *command* objekter, der hovedsagligt gør metoden til et objekt. Følgende klassediagram 5.8 på følgende side viser de forskellige handlinger som betragtes og implementeret som *command objekter*.

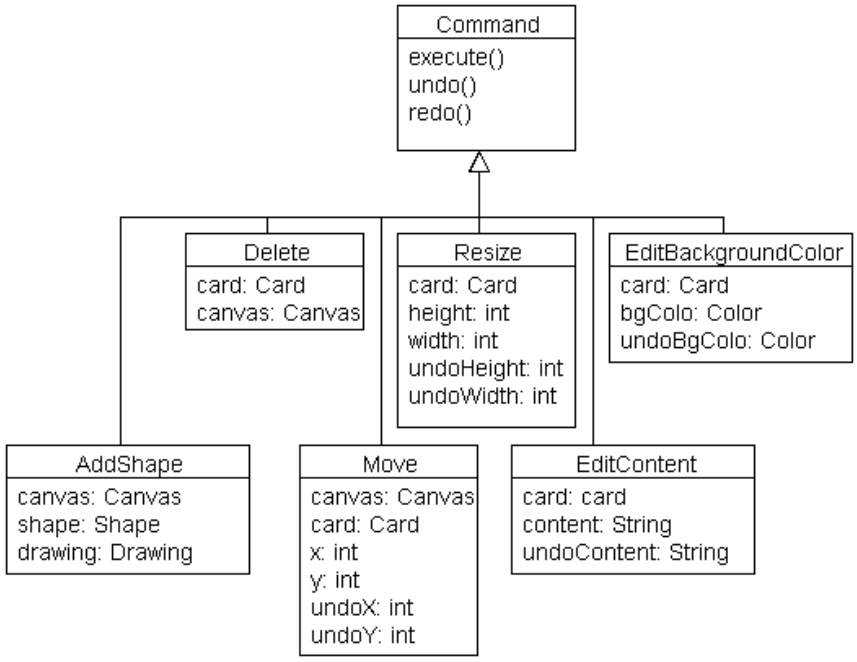
For eksempel, når flytte handling udføres, gemmer objektet de tidligere positioner

```
1 @Override
2 public void execute() {
3         x = comp.getX();
4         y = comp.getY();
5         component.setBounds(getX(),getY(), getWidth(), getHeight());
6 }
```

Listing 5.16: Udfører en handling

Når handlinger fortrydes, gemmes de nuværende positioner inden, for at kunne gentage dem senere

```
1 @Override
2 public void undo() {
3         redoX = component.getX();
4         redoY = component.getY();
```



Figur 5.8: Tegningsform valgmuligheder

```
5             component.setBounds(x, y, getWidth(),getHeight());
6     }
```

Listing 5.17: Fortryder en handling

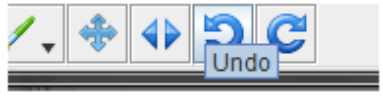
Når en fortrudt handling gentages, genskabes positionen vha. de fortrudte positioner

```
1  @Override
2  public void redo() {
3             component.setBounds(redoX,redoY,getWidth(),getHeight());
4  }
```

Listing 5.18: Gentager en handling

Gentagelsen kan også gøres ved at tilkalde *execute()* metode, men da dette kræver at ændre i koden (at gemme den tidligere position på en anderledes måde), bevares gentagelses metode som det er.

Fortryd og gentag kan udføres fra værktøjslinjen, som kan ses på figur 5.9 som sender beskeden videre til tegningsområdet.



Figur 5.9: Fortryd eller gentage knapper i værktøjslinjen

I princippet kan alle handlinger betragtes som atomare handlinger og implementeres som *command* objekter. Dette er tidskrævende og da dette speciale er tidsbegrænset, har jeg valgt at implementerer kun de førnævnte. Men da mekanismen og datastrukturen er allerede implementeret, vil det være relativt nemt at udvide.

5.11 Visuel feedback

Værktøjet har forskellige tilstande, derfor er det vigtigt at fortælle brugeren, ved hjælp af grafiske elementer, hvilken tilstand der aktiveret i øjeblikket. Værktøjet skal visualisere den nuværende handling, såsom størrelsestilpasning, sletning, flytning eller udvælgelsen. Dette vil som sagt hjælpe brugere med at forstå den nuværende handling og at udføre den ønskede funktion.

1. **Udvælgelse**: Når en tegningslærred eller kort bliver valgt, skal det fremhæves ved at fremhæve dens ramme, for at indikere udvælgelsen. Dette gøres ved at løbe igennem alle de eksisterende tegningslærreder og kort, og fremhæve den udvalgte komponents ramme. Når et andet tegningslærred eller kort bliver udvalgt, skal værktøjet huske at fjerne udvælgelsen af den tidligere udvalgte komponent, således at der er altid højest en enkel udvalgt komponent. Figur 5.10 og 5.11 viser hvordan tegningslærred og kort fremhæves når det vælges med musen.

Figur 5.10: Fremhævning af tegningslærredets kan

Figur 5.11: Fremhævning af kortets kant

2. **Flytning**:

   Når en tegningslærred skal flyttes, ændres musens ikon til den standard flytnings ikon, for at indikere at musen påvirker tegnings position. Musens ikon ændres automatisk når musen kommer over et tegningslærred eller et kort. Følgende kode 5.19 viser hvordan musens ikon ændres:

```
1 this.setCursor(Cursor.getPredefinedCursor(Cursor.MOVE_CURSOR));
```

Listing 5.19: Ændring af musens ikon

3. **Størrelsestilpasning** :

   Når et korts størrelse skal tilpasses, vises kanter rundt omkring kortet, for at hjælpe brugeren visuelt med at tilpasse størrelsen. Dette gøres ved at oprette et nyt objekt som rummer kortet. Objektet *ResizerBorder*, er et

nyt skræddersyet *JComponent* som tegner sig rundt om en komponent. Figur 5.12 viser hvordan *ResizerBorder* objektet tegner sig rundt om kortet, for at indikere at der kan trækkes i kanterne.

Figur 5.12: Størrelses tilpasnings kanter

4. **Sletning** :

   Når et tegningslærred eller kort skal slette, aktiveres slette tilstand som ændrer musens ikon til en viskelæder. Derefter kan komponenten slettes ved at vælge det med viskelæderet. Denne måde er valgt omhyggeligt for at afspejle den samme fremgangsmåde når et objekt flyttes.

5.12 Kildekoden og værktøjet

Kildekoden findes i den vedlagte CD, under kildekode mappen. Kildekoden består af .java filer, og ikoner under src/icons mappen, samt gemte filer i form af XML filer. Kildekoden kan importeres i et Java projekt og derefter køres KanbanTavle.java.

CD'en indeholder også en eksekverbar .jar fil under kildekode mappen også. Jar filen kan køres dirkte derfra.

5.13 Opsummering

Kapitlet viser hvordan whiteboardtavlen kan implementeres. Der er vist hvordan tegningslærreder og tegningsformer kan oprettes, samt kort og dens indhold. Der er også vist hvordan de forskellige tilstande aktiveres og hvad de bruges til. Der blev forklaret hvordan brugerens interaktion fortolkes og behandles af værktøjet, samt hvilke output gives.

Kapitel 6

Test

I dette kapitel vil testen gennemgås, for at vise hvordan systemet blev testet og hvilke resultater testen er kommet frem til. Det vil også komme nærmere på hvilke test strategier der er benyttet.

6.1 Test strategier

Ifølge XP, er det vigtigt at testen udføres for hver udviklings iteration. Dette blev stort set overholdt, på nær små Unit test som ikke kunne implementeres pga. tidspres. Hver iteration blev testet med en automatiserede JUnit test, en automatiserede acceptance test og en manuel test, som hver beskrives yderligere i de næste afsnit. Udover at teste løbende, har jeg testet værktøjet fuldstændigt efter den sidste iteration.

Testen står primært for at teste brugerhistorier for hver iteration. Målet er at bekræfte at funktionaliteten virker som beskrevet, og testen skal bestås for at iterationen kan anses for værende færdig.

Der blev også fortaget en brugervenligheds test for at identificerer hvilke brugervenligheds svagheder værktøjet har, for fremtidige forbedringer.

6.2 Automatiserede test

Den automatiserede Unit test bruges til at teste specifikke dele af koden. Testen udføres for at sikre at koden virker som den skal, også i særlige tilfælde.

Den automatiserede Unit test udføres ved brug af JUnit testen. Testen blev kodet løbende og igennem hver iteration. Der valgtes at begrænse testen til at kun teste modellen. Dette skyldes at hvis modellen virker som den skal, er det nemt at spotte fejl i præsentationen vha. manuel test. Det skyldes også at det er tidskrævende at opstille en JUnit test for GUI'en. Dette kan ses da jeg har forsøgt at kode en JUnit test for GUI'en, som tog utrolig lang tid for at simulere en robot, samt museklik og bevægelse osv.

JUnit testen er oprettet som en ekstra pakke, test, med forskellige test klasser som hver tester den tilsvarende klasse i modellen.

6.3 Acceptance test

Acceptance test verificerer at kundens krav, i form af brugerhistorier, er forstået og implementeret rigtigt. Testen udføres normalt for hver brugerhistorie når den er færdigimplementeret. Testen betegnes også som en blackbox test, som tester de implementerede funktionaliteter imod brugerhistorier, og på baggrund af disse resultater kan der udledes, om funktionaliteten er udviklet rigtigt og hermed er færdig implementeret. Acceptance test er også en automatiserede test, som i følge dette speciale udføres vha. JUnit test.

Den automatiserede acceptance test udføres på modellen, og Acceptance test under bilag B viser de test scenarier der blev beskrevet. Test scenarierne er beskrevet ud fra brugerhistorierne. Testen har hjulpet med at identificere nogle fejl i kode som er blevet rettet. Alle test scenarierne er nu bestået og hermed kan man udlede at funktionaliteterne virker som forventet.

6.4 Manuel test

Manuel tests udføres af en slut brugerne for at sikre at softwaren virker som forventet. Da jeg fravalgte at teste GUI'en vha. automatiserede test, har jeg udført manuel tests for at teste GUI'en.

Manuel testene blev udført i to stadier, igennem hver iteration og i slutning. Hver iteration har et sæt af brugerhistorier som blev testen manuelt og ved interaktionen med GUI'en for at sikre at inputtet blev modtaget og sendt korrekt til modellen, samt at outputtet på GUI'en afspejler modellen. Til sidst, blev værktøjet testet manuelt ifølge acceptance test scenarier. Alle testene er bestået.

6.5 Brugervenlighedstest

En brugervenlighedstest blev udført for at teste brugergrænsefladens anvendelighed. Testen består af spørgsmål som brugere skal besvare, samt opgaver som brugere skal prøve at løse. Testen er en tænke-højt test, hvor brugere skal tænke højt mens opgaven udføres, for at give facilitator en idé om hvordan brugeren forsøger at løse opgaven. Testen bliver udført på baggrund af at jeg har haft kurset *Usability Engineering* ved Rolf Molich. Denne tænke-højt test følger metoden, som beskrevet i bogen Usable Web Design [28].

Testen blev udført ved hjælp af 3 forskellige personer med forskellige IT baggrund og erfaring. Person 1 er 30 år gammel og er uddannet som farmaceut, og har lidt erfaring med Lean. Person 2 er 26 år og studerer Diplom IT på DTU, og har anvendt agile planlægnings værktøjer igennem studiejob. Person 3 er bygningsteknikker studerende og har kendskab til adskillige tegnings værktøjer.

6.5.1 Tests resultater

Ingen af tests personerne har erfaring med Kanban værktøjer. Person 1 og 3 har ikke hørt om Kanban eller Agile. Person 2 har kendskab til Agile og Scrum igennem studiejob og derfor var det relativt nemt at relaterer sig til kanban. Alle personerne har prøvet at tegne med Paint eller andre tegneværktøjer.

Alle personerne havde i starten et problem med at forstå konceptet med at tilføje et tegningslærred. Person 1 havde misforstået ikonet "tilføj tegningslærred"og troede at det er for at starte et helt nyt projekt. Person 2 kunne ikke forstå hvorfor der skulle være flere tegningslærreder end blot et enkelt til hele udviklingen. Person 3 har spurgt ind til tegningslærredet terminologi for at forstå hvordan det hænger sammen. Alle personerne havde det nemt med at finde hvilken tegningsform de skal tegne med, da ikonet siger sig selv. Alle personerne har prøvet med at tegne uden for tegningslærredet område og undrede sig over, at der ikke bliver tegnet. Dette problem skete kun den først gange de prøvede at tegne og efter de opdagede, ved at afprøve, havde de forstået terminologien. Person 1 og

2 nævner at det kræver at man først prøver og forstår værktøjet, og derefter er det nemt at bruge. Person 1 forslog at ændre "tilføj tegningslærred"ikon.

Person 1 forslog at når man vælger at skrive tekst, skal der være markering på tekstens position, som indikerer at der kan indsættes tekst. Person 2 havde svært med at forstå forskellen mellem linje og polylinje og forslog at ændre polylinjens navn. Person 3 havde arbejdet med mange tegnings værktøjer og havde ingen problem med at genkende alle tegningsformer.

Person 1 og 3 er ikke bekendt med post-it kort og hvad de bruges til, og derfor skulle han undersøge de forskellige knapper før han kunne tilføj et kort. Det lykkes at finde tilføj kort knappen som beskrevet af tooltip. Person 1 forslog at skrive tekst sammen med ikonet i værktøjslinjen for at gøre det nemmere at finde indsæt kort knappen. Person 2 havde det ganske nemt med at finde tilføj kort knappen da ikonet efter hans mening siger sig selv.

Person 1 og 3 havde svært med at forstå hvad brugerhistorie er, og efter jeg forklarede at det er blot en note, havde person 1 prøvet af indsætte tekst som tegningsform, ind i kortet. Da dette ikke virkede, prøvede person 1 at højreklikke og til sidst dobbeltklikke. Person 2 og 3 forsøgte at dobbeltklikke efter de ikke kunne finde en indsæt brugerhistorie knap i værktøjslinjen.

Alle personerne havde det forholdsvis nemt med at finde ud af, hvordan tegningslærreder og kort skal flyttes. Person 1 og 3 brugte musen mens tegnetilstand var slået til, og dermed blev der tegnet i stedet for at blive flyttet. Derefter prøvede de at finde flytnings ikon i værktøjslinjen. Person 1 forslog at ændre musens ikon til en pen, når tegnetilstand er slået til, for at indikere at man tegner ligesom den ændres når man flytter. Person 2 ville gerne have muligheden for at flytte tegningsformer inden for afgrænsningsrammen.

Alle personerne havde besvær med at forstå kort tilknytning. Personerne havde også besvær med at forstå hvordan et kort tilknyttes en tegning, og se at kortet er faktisk er tilknyttet. Det var nødvendigt at forklare tilknytnings mekanismen og reglen for personerne, for at de kunne løse opgaven. Efter forklaring lød det logisk for dem, og de havde ingen besvær med at løse opgaven. Person 1 forslog at vise afgrænsningsrammen til den nærmeste tegning således at personen kan se hvor kortet skærer igennem. Person 3 forslog at vise afgrænsningsrammen for alle tegningslærreder når et kort flyttes, og markere det tegningslærred, som kortet skærer, med stiplet kant, for at indikere tilknytning.

Alle personerne har udført sletning uden besvære eller misforståelse. Person 3 forslog at der skal være en mulighed for at slette tegningslærred med de tilhørende kort. Person 1 og 2 forslog også at det skal være muligt at slette en tegningsform. Person 1 forslog at der skal være en mulighed for at undgå at

bekræfte at man vil slette, ved at sætte et "Spørg aldrig igen"flueben.

Person 1 og 3 havde besvære med at finde tilstand for størrelsestilpasning, og det skyldes en fejl i værktøjet, som ændrer musens ikon når musen nærmer sig kortets kanter, selv om tilstanden ikke er slået til. Da dette var et alvorligt problem og rettelsen er meget nem, blev det rettet.

Det har været svært for person 1 og 3 at forstå konceptet med at udvide kortet som en ny tavle. Der var brug for at forklar konceptet før de kunne løse opgaven, som startede med at dobbeltklikke og højreklikke derefter.

Gem og indlæs funktionaliteten gik som forventede, uden besvær, dog glemte alle brugere at gemme filen med filendelsen .xml. Derfor er det en god ide, at tjekke filens typenavn, om den indeholder .xml og tilføje den hvis den ikke gør.

Alle personerne var positivt overrasket for skalering og har bemærket at tegningsformens kvalitet forbliver uanset tavlens størrelse. Person 1 har anbefalet at vise skalerings størrelse mens brugeren skalerer vinduet. Dette kræver en speciel implementeret Listener, da *ComponentListener* kun tilbyder *componentResized*, som bliver kaldt efter vinduets størrelse har ændret sig, når musen slippes.

Som ekstra bemærkning, har Person 2 anbefalet, at kort skulle indeholde små ikoner og knapper til at indsætte ny brugerhistorie eller udvide kortet som vindue osv.

Der har været masser af gode opdagelser og idéer til hvordan værktøjet kan forbedres, brugervenlighedsmæssigt, men pga. den begrænsede tid, nås der desværre ikke at rette disse problemstillinger i dette speciale.

6.6 Opsummering

Kapitlet viser hvordan værktøjet blev testet og hvordan XP hjalp med at teste løbende. Resultatet viser at implementering forgik som planlagt og funktionerne virker som beskrevet i brugerhistorierne. Brugervenlighedstest viste nogle svagheder som kan forbedres og rettes for at opnå en bedre anvendelighed.

Kapitel 7

Evaluering

Projektet har været spændende og lærerigt at arbejde med, især at anvende XP som udviklingsproces. Det har været inspirerende at anvende XP principper og at være åben for løbende ændringer og tilpasninger. Der blev skrevet omkring 4500 linjer kode og omkring 750 linjer test kode. Som mange programmeringsprojekter, bliver der brugt mere tid på programmering end selve rapporten, men XP har hjulpet lidt med at dokumentere en del af udvikling for hver trin. Dette betyder at sammenlagt har jeg ca. brugt 4 måneder på programmering og $1\frac{1}{2}$ måneder på rapportskrivning.

Generelt er jeg positiv indstillet for forløbet og resultaterne, og har overvejet at videreudvikle værktøjet som Open Source i min fritid.

7.1 Udvidelsesmuligheder

Værktøjet mangler nogle funktioner som har stor værdi for brugeren. Dette skyldes den begrænsede tid og de mange funktioner som sådan et værktøj kan indeholde. Derfor de følgende idéer beskriver hvad der kunne tænkes at udvide værktøjet med hvis tiden tillader.

1. **Rich text editor**

   Det er hjælpsomt at formatere brugerhistoriens tekst, så bruger kan opstille formateret dokumenter, samt vedhæft filer, screenshot og indsætte links. En oplagt mulighed at udvide *JEditorPane* og gør brug af *RTF*format.

2. **Parallel redigering**

   Når der er flere brugere som arbejder samtidigt på tavlen, er det vigtigt at tavlen kan synkroniseres mellem brugerne. Dette kan gøres via en simpel server-klient løsning og tavlens tilstand oploades eller sendes igennem en webserver så andre bruger kan hente automatisk. En anden måde at gøre det på, er via peer-to-peer løsning hvor bruger er forbundet til hinanden og tavlens tilstand/ændring sendes mellem brugerne. Denne funktionalitet var oprindeligt en del af planen, men da vi ønsker en form af peer-to-peer løsning, er opgaven for stor til at være med i dette speciale, og det er bedst hvis opgaven udføres i et andet projekt for sig selv.

3. **Zoom ind & ud**

   Da skalering af tavlen er en vigtig del af værktøjet, kan der opstår tilfælde hvor den skalerede tavle er så lille at det er ulæseligt. Derfor at det hjælpsomt med en zoom ind og ud funktion således at der kan zoomes ind i en del af tavlen for at få flere detaljer.

4. **Viske ud**

   Det er hjælpsomt at kunne udviske en hel- eller en del af en tegningsform. At fortryde hele tegningen løser ikke altid problemet.

5. **Analytiske grafer**

   For at effektiviserer processen er det hjælpsomt at generere grafer der viser arbejdsprocessen såsom work-in-progress, flaskehals osv. Dette vil også hjælpe med at få et overblik af hvor meget der er produceret og dermed estimere hvor meget der kan produceres i fremtiden.

6. **Iterations planlægning**

   Da kanban er en agile udviklingsproces, er det vigtigt at kunne planlægge fremtidige iterationer, såvel som at have overblik over forrige iterationer.

7. **Prædefinerede skabeloner**

   Det kan være tidskrævende at opstille og tegne udviklingsfaser. I mange tilfælde, er den klassiske koloner opstilling nok for at komme i gang. Derfor vil det hjælpe brugeren med at komme i gang, at tilbyde prædefineret skabeloner som visualiserer de klassiske udviklingsfaser.

8. **Panel baseret repræsentation af tavlen**

Som vist i analysen, at dele udviklingsfaser i paneler eller koloner, er et udbredt måde at repræsentere Kanban tavlen. Idéen er at vise tavlens tilstand i denne type af præsentation. Dette kan genereres automatisk ud fra tavlens tilstand.

9. **Logs**

   At logge handlinger kan hjælpe med at se hvordan udvikling forgå. Det hjælper også med at se hvem der laver hvad, og hvordan projektet skrider frem.

10. **Oversigt**

    At vise oversigt af tavlen, såsom hvor mange kort der er, hvor mange bliver behandlet/udviklet, hvor mange tegninger osv. Dette giver et hurtigt overblik for manageren.

11. **Konfigurations fil**

    Modellen og Præsentationen indeholder mange forskellige konfigurationer og tekst beskrivelser, som er i øjeblikket hård kodet. En separat konfigurations fil vil hjælpe med at definere tekst strenge og lignende konfigurationer, således at man blot kan ændre den eksternt og uden at redigere .java filerne.

12. **At tegne uden for tegningslærredet**

    Da test brugere har prøvet at tegne uden for tegningslærredets område, er det en god ide at registrere, når brugeren vælger at tegne, uden for tegningslærredet, og dermed oprette et nyt tegningslærred. Dette vil hjælpe brugeren med at tegne uden at tænke så meget over at oprette et nyt tegningslærred for hver udviklingsfase.

13. **Sammenflet/split tegningslærreder**

    For at gøre tegning mere fleksibel, er det en god ide at tillade sammenfletning af to eller flere tegningslærreder. På samme måde når der er et stort tegningslærred med mange tegningsformer, er det også hjælpsomt at tillade at splitte det i to eller flere tegningslærreder. Dette vil også betyder at man nemt og hurtigt kan flytte tegningsformer fra det ene tegningslærred til et andet.

14. **Genvejtaste**

    Det vil være hurtigere at udføre handlinger, hvis der er genvejtaste til værktøjslinjen. Det kan for eksempel være at fortryde ved at klikke ctrl + z, gendanne ved ctrl + y eller at gemme ved ctrl + s osv.

Kapitel 8

Konklusion

I denne rapport er det vist hvor vigtig whiteboardtavle er, og hvor udbredt det er i nuværende. Whiteboardtavlen er en vigtig hjælpemiddel i vores hver dag, og sammen med post-it kort kan være en stor hjælp til at planlægge og udføre opgaver.

Undersøgelser viser også at agile udviklingsprocesser er udbredt, og mange projekter visualiserer udviklingsprocessen vha. whiteboardtavler og post-it kort, hvor en af mulighederne er Kanban tavlen. På trods der allerede findes mange Kanban værktøjer, er der identificeret mangel på grundlæggende funktionaliteter såsom at tegne, at skalere og at udvide kort som tavler.

Resultatet af dette speciale er, et unikt værktøj, der simulerer en whiteboardtavle, som understøtter at tegne med forskellige tegningsformer, og at oprette kort. Tavlen er skalerbar og kan anvendes i forskellige skærmeopløsninger, da den kan tilpasse sig ved at skalere indholdet, uden at miste kvalitet eller overblikket. Kort kan indeholde en note, eller kan udvides som en nye tavle, hvilket betyder at det er muligt for et kort at repræsentere et helt projekt.

Værktøjet kan gemmes og indlæses ved et givent tidspunkt, nemt og hurtigt. En række af handlinger i værktøjet er implementeret som atomare funktioner, som betyder at det har været muligt at implementere en fortrydelses og gentagelses funktionalitet.

Derudover er det muligt at tilføje, slette, flytte og tilpasse planlægnings objekterne (tegningslærreder og kort). Der er i rapporten vist hvordan værktøjet er designet, og hvilke designvalg og principper blive anvendt og hvorfor. Der er også vist hvordan implementering forgik og hvad værktøjets dele består af.

Der er vist at det er muligt at udvikle et unikt værktøj der tager højde til en række af udvidelser som ikke findes i andre værktøjer. Værktøjet kan tilpasses for at understøtte kanban tavle i softwareudvikling, men kan også tilpasses for at understøtte mange andre sammenhænge.

Litteratur

[1] Cox Donald, *Supporting Collaborative Interpretation in Distributed Groupware.* Addison Wesley, Massachusetts, 2nd Edition, 1994: [online] Tilgngeligt: http://grouplab.cpsc.ucalgary.ca/grouplab/uploads/Publications/Publications/2000-CollabInterp.CSCW.pdf [hentede den. 15. august 2011]

[2] Cox Donald, *Supporting Results Synthesis in Heuristic.* University of Calgary, 1998, [online] Tilgngeligt: http://grouplab.cpsc.ucalgary.ca/grouplab/uploads/Publications/Publications/1998-Cox.MScThesis.pdf [hentede den. 15. august 2011]

[3] John Wiley & Sons *Kanban.* Principles of Computer-Integrated Manufacturing, 1992

[4] Ohno, Taiichi *Toyota Production System - beyond large-scale production.* Productivity Press, 1988

[5] Scott Ambler *Agile Adoption Survey.* 2007, [online] Tilgngeligt: http://drdobbs.com/architecture-and-design/200001986?pgno=1 [hentede den. 15. august 2011]

[6] Six Sigma CMMI *Extreme Programming (XP)* . [online] Tilgngeligt: http://www.sei.cmu.edu/library/assets/jarvis-gristock.pdf [hentede den. 15. august 2011]

[7] Don Wells *Extreme Programming: A gentle introduction.* 2009, [online] Tilgngeligt: http://www.extremeprogramming.org [hentede den. 15. august 2011]

[8] Martin Fowler *Chrysler Comprehensive Compensation System.* [online] Tilgngeligt: http://www.martinfowler.com/bliki/C3.html [hentede den. 15. august 2011]

[9] Rod Hutchings *Timeboxing.* 2008 [online] Tilgngeligt: http://www.projectmanagement.net.au/triple_constraints [hentede den. 15. august 2011]

[10] Don Wells *Extreme Programming: A gentle introduction.* [online] Tilgngeligt: http://www.extremeprogramming.org/rules/early.html [hentede den. 15. august 2011]

[11] Xin Wang, Frank Maurer, Robert Morgan and Josyleuda Oliveira *Tools for Supporting Distributed Agile Project Planning.* [online] http://ase.cpsc.ucalgary.ca/uploads/Publications/Wangetal2010.pdf [hentede den. 15. august 2011]

[12] Amy Fowler *A Swing Architecture Overview.* [online] Tilgngeligt: http://java.sun.com/products/jfc/tsc/articles/architecture/ [hentede den. 15. august 2011]

[13] Java *Abstract Window Toolkit.* [online] http://java.sun.com/products/jdk/awt/ [hentede den. 15. august 2011]

[14] Sharon Zakhour and Anthony Petrov *Mixing Heavyweight and Lightweight Components.* 2009, [online] http://java.sun.com/developer/technicalArticles/GUI/mixing_components/ [hentede den. 15. august 2011]

[15] Hansen, Kristian *Avanceret Java-programmering.* [online] http://www.avanceret-java-programmering.dk/ [hentede den. 15. august 2011]

[16] *Java Event Listeners..* [online] http://download.oracle.com/javase/tutorial/uiswing/events/index. [hentede den. 15. august 2011]

[17] Steve Burbeck *How to use Model View Controller* [online] http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html [hentede den. 15. august 2011]

[18] *JavaBeans Concepts.* [online] http://download.oracle.com/javase/tutorial/javabeans/whatis/inde [hentede den. 15. august 2011]

[19] *Minimum Bounding Box.* [online] http://en.wikipedia.org/wiki/Minimum_bounding_box [hentede den. 15. august 2011]

[20] Stochmailek, Lech Madeyski og Michael *Architetural Design of Modern Web Applications*. [online] http://www.fatih.edu.tr/ moktay/2007/fall/ceng401/madeyski05architectural.pdf [hentede den. 15. august 2011]

[21] Roberts, Eric *Designing a Java graphics library*. [online] http://cs.stanford.edu/ eroberts/papers/ITiCSE-1998/JavaGraphicsLibrary.pdf [hentede den. 15. august 2011]

[22] *Billede format*. [online] http://www.abmedia.dk/video/format.htm [hentede den. 15. august 2011]

[23] Amy Fowler *Painting in AWT and Swing*. [online] http://java.sun.com/products/jfc/tsc/articles/painting/ [hentede den. 15. august 2011]

[24] Nordfalk, Jacon *Javabog Kapitel 18 Serialisering af objekter*. [online] http://javabog.dk/OOP/kapitel18.jsp [hentede den. 15. august 2011]

[25] Bala Paranj *how to implement the Command pattern in Java*. 1999, [online] http://www.javaworld.com/javaworld/javatips/jw_javatip68.html [hentede den. 15. august 2011]

[26] Matthew Heaney *Memento pattern*. [online] http://adapower.com/index.php?Command=Class&ClassID=Patterns&CID=271 [hentede den. 15. august 2011]

[27] *Mouse Motion Listener*. [online] http://download.oracle.com/javase/tutorial/uiswing/events/ [hentede den. 15. august 2011]

[28] Molich, Rolf *Usable Web Design*. Kapitel 11 s. 128

Bilag A

Use cases

Use Case	Tegn på tavlen
Målsætning	At tegne en tegning på tavlen
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan tegne frit med musen
Prækondition	Der er oprettet et tegningslærred
Postkondition	Tegning er tegnet på tavlen
Trigger	Bruger vælger at tegne på tavlen
Scenariet	1. Oprette et tegningslærred 2. Vælger at aktivere tegnings tilstand på tegningslærredet via værktøjslinje 3. Bruger musen til at tegne frit i det valgte tegningslærred.

Use Case	Flytte tegninger
Målsætning	At flytte en tegning rundt på tavlen
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan flytte tegninger med musen
Prækondition	Der er oprettet et tegningslærred
Postkondition	Tegning er flyttet til den ønskede position
Trigger	Bruger vælger at flytte et tegningslærred på tavlen
Scenariet	1. Vælger en tegning 2. Bruger musen til at tegne frit i det valgte tegningslærred

Use Case	Skalere tegningslærred
Målsætning	At skalere tegningslærred
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan skalere tavlen og dens indhold
Prækondition	• Tavlen er allerede åben • Der er oprettet et tegningslærred
Postkondition	Tegning er flyttet til den ønskede position
Trigger	Bruger vælger at flytte et tegningslærred på tavlen
Scenariet	1. Vælger en tegning 2. Bruger musen til at tegne frit i det valgte tegningslærred

Use Case	Tilføje kort
Målsætning	At tilføje kort
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan tilføje nye kort
Prækondition	Tavlen er allerede åben
Postkondition	Et nyt kort er oprettet
Trigger	Bruger vælger at tilføje et nyt kort
Scenariet	1. Oprette et nyt kort

Use Case	Flytte kort
Målsætning	At flytte kort på tavlen
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan flytte kort
Prækondition	• Tavlen er allerede åben • Der er oprettet et kort
Postkondition	Et nyt kort er flyttet
Trigger	Bruger vælger at flytte et kort
Scenariet	1. Vælge kortet med musen 2. Flytte kortet til den ønskede position på tavlen

Use Case	Tilknytte kort
Målsætning	At knytte kort til en tegning
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan knytte kort til tegninger
Prækondition	• Tavlen er allerede åben • Der er oprettet et eller flere tegningslærreder
Postkondition	Kort er tilknyttet
Trigger	Bruger vælger at tilknytte kort med tegningslærred
Scenariet	1. Oprette et nyt kort 2. Flytte kortet således at det skære på den ønskede tegningslærred 3. Kortet tilknyttes automatisk til den tætteste tegning

Use Case	Kort følger med tegningslærred
Målsætning	Korts position flyttes sammen med den tilknyttede tegningslærred
Aktør	Bruger af værktøjet
Beskrivelse	Kort flyttes automatisk når den tilknyttede tegninger flyttes
Prækondition	• Tavlen er allerede åben • Der er oprettet et eller flere tegningslærreder • Der er oprettet kort • Der er tilknyttet kort til tegninger
Postkondition	Kort er flyttet
Trigger	Bruger vælger at flytte tegningslærredet med de tilknyttede kort
Scenariet	1. Flyt en tegning som har tilknyttede kort

Use Case	Kort indhold
Målsætning	At tilføje brugerhistorie til kort
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan tilføje brugerhistorie til kort
Prækondition	• Tavlen er allerede åben • Der er oprettet kort
Postkondition	Brugerhistorien er tilføjet til kortet
Trigger	Bruger dubletklikker på et kort
Scenariet	1. Dubletklike et kort 2. Indtast brugerhistorien i form af tekst

Use Case	At gemme
Målsætning	At gemme tavlens tilstand
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan gemme tavlens tilstand
Prækondition	Tavlen er allerede åben
Postkondition	Tavlen er gemt
Trigger	Bruger vælger at gemme
Scenariet	1. Klik på gem knappen 2. Indtast fil navn 3. Tavlen er gemt i form af en xml fil

Use Case	At loade
Målsætning	At indlæse tavlens tilstand
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan indlæse en allerede udarbejdet tavle
Prækondition	Tavlen er allerede gemt som en fil
Postkondition	Tavlen er indlæst
Trigger	Bruger vælger at indlæse
Scenariet	1. Klike på load knappen 2. Vælge filen 3. Tavlen er indlæst

Use Case	Understøttelse af forskellige tegningsformer
Målsætning	Understøttelse af forskellige tegningsformer
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan tegne forskellige former
Prækondition	Der er oprettet et tegningslærred
Postkondition	Tegningsformen er tegnet på tavlen
Trigger	Bruger vælger at tegne på tavlen
Scenariet	1. Vælge et tegningsform 2. Bruge musen til at tegne i det valgte tegnings-lærred.

Use Case	Skalere tavlen
Målsætning	At skalere tavlen
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan skalere tavlen
Prækondition	• Tavlen er allerede åben • Der er oprettet et tegningslærred og kort
Postkondition	Tavlen samt tegninger og kort er skaleret
Trigger	Bruger dubletklikker på et kort
Scenariet	1. Skalere tavlens størrelse

Use Case	Slette tegningslærred
Målsætning	At slette tegningslærred
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan slette tegningslærred
Prækondition	Der er oprettet nogle tegningslærreder
Postkondition	Det valgte tegningslærred er slettet
Trigger	Bruger vælger at slette et tegningslærred
Scenariet	1. Aktivere slette tilstanden 2. Slette tegningslærred ved at vælge det med musen

Use Case	Slette kort
Målsætning	At slette kort
Aktør	Brugeren kan slette kort
Beskrivelse	Brugeren kan slette tegningslærred
Prækondition	Det er allerede oprette nogle kort
Postkondition	Det valgte kort er slettet
Trigger	Bruger vælger at slette et allerede eksisterende kort
Scenariet	1. Aktivere slette tilstanden 2. Slette kort ved at vælge det med musen

Use Case	At tegne i forskellige tavle størrelser
Målsætning	At understøtte tegning i forskellige skalerings størrelser
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan tegne selvom tavlen er skaleret
Prækondition	• Der er oprettet et tegningslærred • Tavlen er skaleret
Postkondition	Tavlen samt tegninger og kort er skaleret
Trigger	Bruger dubletklikker på et kort
Scenariet	1. Skalere tavlen 2. Tegn med musen

Use Case	Tegningsformer
Målsætning	At understøtte flere tegningsformer
Aktør	Brugeren kan slette kort
Beskrivelse	Brugeren kan tegne med forskellige tegningsformer
Prækondition	Der er oprettet et tegningslærred
Postkondition	Tegningsformen bliver tegnet
Trigger	Bruger vælger at tegne
Scenariet	1. Vælge en tegningsform 2. Tegne med musen

Use Case	Visuel feedback
Målsætning	At give visuel feedback
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren får en visuel feedback når tegninger tilpasses eller flyttes
Prækondition	Der er oprettet et tegningslærred
Postkondition	Visuel feedback er præsenteret i GUI'en
Trigger	Bruger vælger at tilpasse størrelse eller flytte en tegningslærred
Scenariet	1. Vælge en tegningslærred 2. Vælge at tilpasse eller flytte tegningen

Use Case	Kort hierarki
Målsætning	At udvide kort som en ny tavle
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan udvide kort som tavle
Prækondition	Der er oprettet nogle kort
Postkondition	Visuel feedback er præsenteret i GUI'en
Trigger	Bruger vælger at tilpasse størrelse eller flytte en tegningslærred
Scenariet	1. Vælge et kort med musen 2. Udivde et kort som en tavle ved at højreklikke og vælge "Open Board"

Use Case	Korts baggrund
Målsætning	At ændre korts baggrund
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan ændre korts baggrund
Prækondition	Der er oprettet nogle kort
Postkondition	Korts baggrund ændres
Trigger	Bruger vælger at ændre korts baggrund
Scenariet	1. Højreklike på et kort 2. Vælge "Change Background" 3. Vælge den ønskede farve 4. Klikke "Select Color"

Use Case	Undo/Redo
Målsætning	At fortryde eller gentage en handling
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan fortryde eller gentage en handling
Prækondition	• Der er oprettet nogle tegningslærred • Der er tegnet nogle tegningsformer • der er oprettet kort og udført andre handlinger
Postkondition	Handlingen bliver fortrudt eller gentaget
Trigger	Bruger vælger at fortryde eller gentage en handling
Scenariet	1. Tegne forskellige tegningsformer 2. Flytte tegningsformer og kort 3. Indsæt brugerhistorie 4. Ændre korts baggrundsfarve 5. Fortryd og gentage handlinger ved at trykke på "Undo"og "Redo"i værktøjeslinjen

Use Case	Tekst redigering
Målsætning	At redigere en allerede eksisterende tekst
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan redigere en allerede oprettet tekst.
Prækondition	• Der er oprettet nogle tegningslær-red • Der er allerede oprettet en tekst
Postkondition	Teksten er redigeret
Trigger	Bruger vælger at redigerings teksten
Scenariet	1. Vælge "Edit-¿ "Edit Text" 2. Vælge tekst der ønskes at ændres 3. Redigere teksten ved at taste nye tekst ind 4. Fortryd og gentage handlinger ved at trykke på "Undo"og "Redo"i værktøjeslinjen

Use Case	Tegningsfarver
Målsætning	At tegne med farver
Aktør	Bruger af værktøjet
Beskrivelse	Brugeren kan tegne med farver
Prækondition	Der er oprettet et tegningslærred
Postkondition	Visuel Tegningsformen er tegnet med den ønskede farve
Trigger	Bruger vælger at ændre tegningsfarve
Scenariet	1. Vælge "Color" 2. Vælge den ønskede farve 3. Klikke "Select Color"

Bilag B

Acceptance test

1	**Tilføje tegningslærred**			
Formål: Brugeren skal kunne oprette et nyt tegningslærred for at tegne på.				
Forudsætninger: Tavlen er åben.				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Klik på "Add New Canvas"knappen	Et tegningslærred er oprettet	x	

2	**Tegne på tegningslærredet**			
Formål: Brugeren skal kunne tegne på tegningslærredet.				
Forudsætninger: Tavlen er åben.				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Klik på "Add New Canvas"knappen	Et tegningslærred er oprettet	x	
2	Klik på "Enable drawing"knappen	Tegningstilstand er aktiveret	x	
3	Brug musen til at tegne	Musen tegner de ønskede former	x	

3	**Flytte tegningslærredet**			
Formål: Brugeren skal kunne flytte tegningslærredet.				
Forudsætninger: Tavlen er åben. Et eller flere tegningslærreder er tilføjet				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Klik på "Enable moving"knappen	Flytningstilstand er aktiveret	x	
2	Brug musen til at flytte tegningslærred	Tegningslærredet er flyttet til den ønskede position	x	

4	**Tilpasse tegningslærredets størrelse**			
Formål: Brugeren skal kunne tilpasse tegningslærredets størrelse				
Forudsætninger: Tavlen er åben.				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Klik på "Enable resizing"knappen	Størrelsestilpasningstilstand er aktiveret	x	
2	Brug musen til at tilpasse tegningslærredets størrelse	Tegningslærredets størrelse er tilpasset.	x	

6	**Tilføje kort**			
Formål: Brugeren skal kunne flytte kort.				
Forudsætninger: Tavlen er åben. Et eller flere kort er tilføjet.				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Brug musen til at flytte kort	Kort er flyttet til den ønskede position	x	

7	**Tilføje brugerhistorie**			
Formål: Brugeren skal kunne tilføje en brugerhistorie til kort.				
Forudsætninger: Tavlen er åben. Et eller flere kort er tilføjet.				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Dobbeltklikke på et kort	En nyt vindue åbnes for at indtaste brugerhistorien	x	
2	Indtast brugerhistorien og klik OK	Brugerhistorien er gemt og vist i kortet	x	

8	**Tilknytte kort med tegningslærred**			
Formål: Brugeren skal kunne tilknytte kort med et tegningslærred.				
Forudsætninger: Tavlen er åben. Et eller flere tegningslærreder og kort er tilføjet.				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Flyt kortet nærmere på tegningen således at det skære afgrænsningsrammen	Kortet er tilknyttet den nærmeste tegning	x	

9	**Kort flyttes sammen med tegningslærred**			
Formål: Brugeren skal kunne flytte et tegningslærred og dens tilhørende kort ved at blot flytte tegningslærredet.				
Forudsætninger: Tavlen er åben. Et eller flere tegningslærreder og kort er tilføjet.				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Flytte tegningslærredet	Kort følge med tegningslærredet	x	

10	**Gemme**			
Formål: Brugeren skal kunne gemme tavlens tilstand.				
Forudsætninger: Tavlen er åben. Et eller flere tegningslærreder og kort er tilføjet.				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Tryk på "Save"knappen	Nyt vindue åbnes for at indtaste filens navn	x	
2	Indtast filens navn	Fils navn indtastet	x	
3	Tryk OK	Filen er gemt	x	

11	**Indlæse**			
Formål: Brugeren skal kunne indlæse tavlens tilstand fra en allerede gemt fil.				
Forudsætninger: Tavlen er åben. Ingen tegninger eller kort oprettet.				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Tryk på "Load"knappen	Nyt vindue åbnes for at vælge den gemte filen	x	
2	Vælg filen	Filen er valgt	x	
3	Tryk OK	Filen er indlæst	x	

12	Tegningsformer			
Formål: Brugeren skal kunne tegne forskellige tegningsformer.				
Forudsætninger: Tavlen er åben.				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Tryk på "Add Canvas"knappen	Et nyt tegningslærred er oprettet	x	
2	Højreklik på tegningslærred og vælg en tegningsform	Højreklik menuen vises	x	
3	Tegn vha. musen	Den ønskede tegningsform tegnes	x	
4	Gentage 2-3 med forskellige tegningsformer	De ønskede tegningsformer tegnes	x	

13	Automatisk tilpasning af tegningslærredet			
Formål: Brugeren skal kunne skalere tavlen og alle dens tegninger og kort.				
Forudsætninger: Tavlen er åben. Der er oprettet tegningslærred og kort				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Gør vinduet mindre/større vha. musen	Vinduets samt tegninger og kort skaleres i forhold til størrelsen	x	

14	Skalering			
Formål: Brugeren skal skalere tavlen og alle dens tegninger og kort.				
Forudsætninger: Tavlen er åben. Der er oprettet tegningslærreder og kort				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Gør vinduet mindre/større vha. musen	Vinduets samt tegninger og kort skaleres i forhold til størrelsen	x	

15	Slette tegninger			
Formål: Bruger skal kunne slette allerede eksisterende tegning.				
Forudsætninger: Tavlen er åben. Der er oprettet tegningslærred				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Aktivere slette tilstand	Slette tilstand er aktiveret.	x	
1	Klikke på det tegningslærred der ønskes slettet	Tegningslærreder slettet	x	

16	Slette kort			
Formål: Bruger skal kunne slette allerede eksisterende kort.				
Forudsætninger: Tavlen er åben. Der er oprettet kort				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Aktivere slette tilstand	Slette tilstand er aktiveret.	x	
1	Klikke på det kort der ønskes slettet	kort er slettet	x	

17	Visual feedback			
Formål: Bruger skal kunne se en visuel feedback når der vælges at flytte eller tilpasse størrelsen af en tegning.				
Forudsætninger: Tavlen er åben. Der er oprettet tegningslærreder og kort				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	vælge et tegningslærred eller kort	Komponenten er valgt og hævede	x	

18	Kort hierarki			
Formål: Bruger skal kunne udvide et kort som en nye tavle.				
Forudsætninger: Tavlen er åben. Der er oprettet kort				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Højreklikke på et kort og vælg "Open Board"	Nye tavle er åbnet	x	

18	Kort hierarki			
Formål: Bruger skal kunne udvide et kort som en nye tavle.				
Forudsætninger: Tavlen er åben. Der er oprettet kort				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Højreklikke på et kort og vælg "Open Board"	Nye tavle er åbnet	x	

19	**Tegningsfarver**			
Formål: Bruger skal kunne tegne med farve.				
Forudsætninger: Tavlen er åben. Der er oprettet et tegningslærred				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Højreklikke på tegningslææredet og vælge "Color"	Et nyt vindue åbnes, hvor der kan vælges farver	x	
2	Vælg den ønskede farve	Farven er valgt	x	
3	Tegn en tegnings-form	Den ønskede tegningsform er tegnet med den ønskede farve	x	

20	**Korts baggrundsfarve**			
Formål: Bruger skal kunne definere korts baggrundsfarve				
Forudsætninger: Tavlen er åben. Der er oprettet et kort				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Højreklikke kort og vælge "Background Color"	Et nyt vindue åbnes, hvor der kan vælges farver	x	
2	Vælg den ønskede farve	Farven er valgt	x	
3	Klik OK	Baggrundsfarve er sat	x	

21	**Redigere tekst**			
Formål: Bruger skal kunne redigere tekst				
Forudsætninger: Tavlen er åben. Der er oprettet tekst				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Højreklikke på tegningslærredet og vælge "Edit-¿ "Edit text"	Tekstredigerings tilstand er aktiveret	x	
2	Vælge teksten med musen	Teksten er valgt	x	
3	Redigere teksten	Teksten er redigeret	x	

22	**Fortryd**			
Formål: Bruger skal kunne fortryde en handling				
Forudsætninger: Tavlen er åben. Der er oprettet tegningslærreder og kort				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Udføre nogle atomare handlinger	Handlingen er udført	x	
2	Fortryde handlingen ved at klikke på "Undo"knappen	Handlingen er fortrudt	x	

23	**Gentage**			
Formål: Bruger skal kunne gentage en handling				
Forudsætninger: Tavlen er åben. Der er oprettet tegningslærreder og kort				
	Trin	**Forventet**	**Bestået**	**Mislykkes**
1	Fortryde handlinger	Handlingen er fortrudt	x	
2	Gentage handlingen ved at klikke på "Redo"knappen	Handlingen er gentaget	x	

Bilag C

Brugervenlighedstest

C.1 Prtest

Programmet skal vre startet p en Windows maskine. Alle gemte XML filer er slettet. Ingen tidligere tegninger eller kort skal vre p tavlen.

C.2 Prtest interview

Hej og velkommen. Tak fordi du vil hjlpe os med at teste bruger venligheden af vrktjet Kanban tavle.

Jeg vil gerne gre opmrksom p at det p ingen mde er en test af dig men alts en test af vrktjet. Med andre ord kan du ikke gre noget forkert under denne test. Jeg vil forholde mig neutral, s give gerne sp mange kommentar som muligt.

Jeg undskylder p forhnd at jeg kun vil svare p f sprgsml fra din side af under testen. Sprg alligevel da dine sprgsml kan give mig meget vigtige ledetrde. Jeg stiller mske underlige sprgsml. Grunden til dette er at jeg gerne vil forst hvordan du fortolker dele af vrktjet.

Testen af vrktjet vil foreg p den mde at du bliver stillet en rkke opgaver du skal udfre, mens du tnker hjt.

Inden vi gr i gang vil jeg hre lidt om din baggrund. Du vil forblive anonym.

- Hvad er din uddannelse
- Hvor gammel er du
- Har du hrt om Agile? Kanban?
- Har du prvet Kanban vrktjer fr?
- Har du prvet Agile planlgnings vrktjer?
- Har du brugt paint eller lignende tegneprogrammer?

C.3

1. Du er ansat i et nyt firma og skal vre en del af produktionen, derfor er du adspurgt at lre at bruge vrktjet. Start med tegne et vilkrligt tegn, en linje, en firkant, en cirkel og tekst.
2. Indst et nyt kort.
3. Indst en brugerhistorie i kortet.
4. Tilknyt kortet til tegning
5. Tegn en ny tegning
6. Flyt den ene tegning yderst til venstre og den anden yderst til hjre
7. Tilfj nyt kort og tilknyt det til den anden tegning
8. Slet en tegning og dens tilhrende kort
9. Tilfj et nyt kort og tilpas strrelsen
10. Udvide det sidste oprettet kort som tavle
11. Gem tavlen og luk programmet
12. Start programmet igen og load den gemte fil.
13. Skaler tavlen og observer hvordan tegninger ndrer sig