Institutionen för datavetenskap
Department of Computer and Information Science

Examensarbete

Metoder för användardriven gränssnittsprogrammering

av

Jonathan Crusoe

LIU-IDA/LITH-EX-G--14/047--SE

2014-06-09

Linköpings universitet

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Linköpings universitet
Institutionen för datavetenskap

Examensarbete

Metoder för användardriven gränssnittsprogrammering

av

Jonathan Crusoe

LIU-IDA/LITH-EX-G--14/047--SE

2014-06-09

Handledare: Klas Arvidson

Examinator: Jonas Wallgren

Linköpings universitet
Institutionen för datavetenskap

Innehållsförteckning

Metoder för användardriven gränssnittsprogrammering

Jonathan Crusoe

SAMMANFATTNING

När användare bestämmer sig för att utveckla gränssnitt till sina system sker detta via någon form av verktyg. Vi måste avgöra vilken utvecklingsmetodik som ska användas och hur vi kan tillföra mer funktionalitet för att systemet inte ska bli föråldrat. För att svara på detta bryter vi upp arbetet i två delar. I första delen undersöker vi vilken programmeringsmetodik som lämpar sig bäst för gränssnittsutveckling genom en undersökning i två delar. I andra delen ser vi över vilka lösningar som existerar för att implementera ny funktionalitet till ett verktyg för att sedan presentera en egen lösning.

INLEDNING

2.1 Motivering

När en person utan erfarenhet i mjukvaruutveckling vill utveckla ett grafiskt gränssnitt via ett verktyg, vilken programmeringsmetodik är då bäst att använda? Vi kan argumentera för textuell programmering eftersom det är den metod vi har lärt oss i utbildningen. Kanske finns det bättre metoder för att närma sig problemet för att finna en enklare lösning? Om vi tar fram en programmeringsmetodik kommer användaren förr eller senare vilja ha mer funktionalitet. Då måste vi med någon metod kunna implementera funktionalitet för att uppnå hög utvecklingsbarhet. Om inte detta uppnås kommer användaren att förskjuta verktyget med tiden och införskaffa ny mjukvara.

2.2 Syfte

Ovan nämner vi att rapporten är uppdelad i två delar. Varje del svarar på en frågeställning som nämns nedan. Vi börjar med att genomföra en undersökning i två delar, där vi för varje del implementerar en prototyp. Ifrån den sista prototypen kan vi se vad som behöver vara utvecklingsbart. Med detta kommer vi över till andra delen av rapporten. Här tar vi upp begreppet anveckling och tre olika metoder för hur dessa medger utvecklingsbarhet. Utifrån detta presenterar vi en arkitektur som använder en av metoderna.

2.3 Frågeställning

- Vilket av visuell, textuell och modulär programmeringsmetodik lämpar sig bäst för grafisk gränssnittsutveckling?
- Hur kan en modulär arkitektur realiseras så att systemet medger hög utvecklingsbarhet?

2.4 Avgränsningar

Vi kommer att avgränsa hur många deltagare som är med i undersökningen på grund av brist på tid och resurser även hur många designiterationer som sker på samma grund. Programmeringsmetodikerna som kommer att undersökas i denna rapport är visuell, textuell och modulär.

BAKGRUND

Katrineholms äventyrsklubb (KÄK) anordnar olika evenemang. Bland klubbens största evenemang är Rex, vilket är ett postapokalyptiskt lajv. Evenemanget utspelar sig i en stängd lokal. Du som deltagare har en fiktiv roll som är självskapad. Under lajvet används Terminalen(klient) för att sända och ta mot mejl för att simulera en värld utanför. Eftersom det är teaterspel och arrangörerna vill ge en känsla av verklighet vill de kunna skapa egna sidor för klienten och även kunna svara på mejl. Arrangörerna har en vision för Terminalen, men inte verktyg och kunskap för att uppnå drömmen. Tidigare problem med Terminalen har varit; låg användbarhet både för användare och utvecklare, utvecklingsbarhet, inte uppnått de krav som ställts på utseende och en del av Terminalerna har bara varit tillfälliga fasader tills något bättre presenterats (KÄK använde kort ett system där de hade två mappar på skrivbordet i Windows 95 miljö).

Det finns få i KÄK som är utbildade för mjukvaruutveckling och de som är det vill inte ta in arbetslivet i sin hobby därför behöver de ett verktyg för att lätt utveckla Terminalen.

Även om vi har verktyget så kan vi inte förutspå KÄKs framtida behov av funktionalitet. Att bara ge föreningen källkod och instruktioner för vidareutveckling krockar med tidigare påstående. Detta leder till att vi måste presentera en lösning för att nå hög utvecklingsbarhet av systemet som de kan använda. Vi kommer i rapporten benämna detta verktyg som Verktyget.

TEORI

4.1 Visuell, Textuell eller Modulär

Nedan beskriver vi visuell, textuell och modulär programmeringsmetodik. För att klargöra så ser vi visuell och textuell som motsatser där modulär hamnar i mitten vilket ger metodiken drag ifrån båda. Vi använder flödesprogrammering och därför kommer metodikerna basera sig på detta.

4.1.1 Flödesprogrammering

Hils, Daniel D[18] skriver att det centrala konceptet för flödesmodellen är att vi kan representera ett program som en riktad graf, där noderna representerar funktioner och kanterna flödet av data mellan noder. En kant som lämnar en nod representerar utgående data och en som inträder representerar ingående data. Om vi har noderna A och B med en kant som går från A till B säger vi att data flödar ifrån A till B, samt att A är den övre noden och B den undre noden. Vidare beskriver Hils två olika exekveringsmetoder: datadriven och efterfrågedriven.

Vid datadriven exekvering körs en nod när data anländer på en ingångskant. Den bearbetar datan som sen sänds vidare genom dess utgångskanter. Detta leder till att en nod exekverar så fort data blir tillgänglig och att

informationen alltid kommer att flöda nedåt.

Vid efterfrågedriven exekvering körs en nod endast om data förfrågas. All data flyter fortfarande nedåt men alla efterfrågningar flyter uppåt. T.ex. en display som vill vissa någon form av data.

En slutsats vi kan dra från andra språk (Se 4.1.3) är att vi kan blanda data- och efterfrågedriven exekvering på så sätt att vi får ”semidriven” exekvering. Vi antar ett scenario där vi har noderna Knapp, Inloggning, FältA, FältB och Popup. Kanterna går; Knapp → Inloggning, FältA → Inloggning, FältB → Inloggning och Inloggning → Popup. Användaren aktiverar Knapp med ett ”musklick” och kommer då via datadriven exekvering aktivera Inloggning. Här byter vi över till efterfrågedriven exekvering och aktiverar FältA och FältB så att data från fälten flödar till Inloggning. Vi bearbetar datan i noden och byter över till datadriven exekvering för att aktivera Popup, som vid aktivering visar någon form av text. Metodikerna baserar sig på detta genom att noder kommer att vara objekt som kan vara grafiska. Grafiska objekt kan vara knappar, texter, fält eller listor. Icke-grafiska objekt kan vara sända/hämta data till/från server/klient, byte av nuvarande Terminalsida eller visa ett meddelande för användaren. Kanterna används för kommunikation mellan objekt (t.ex. knapp som visar text).

4.1.2 Textuell programmeringsmetodik

När vi idag använder Internet sker detta via någon form av webbläsare som tolkar HTML-filer[15] till ett grafiskt gränssnitt. Detta gör märkspråket väldigt attraktivt då det baserar sig på att representera ett grafiskt gränssnitt i form av ett dokument. Däremot följer det en standard som inte är anpassad för Terminalen. Ett friare märkspråk är XML[16], där vi lätt kan definiera egna taggar. Det är likt HTML – I båda märkspråken har vi element som börjar (<tagg>) och slutar (</tagg>) med tagg.

Styrkan och svagheten med att använda XML är att det redan existerar tolkar[17] men att vi måste bestämma hur våra taggar ska uppfattas och användas. Vi kommer att använda oss av Figur 1 vid förklaringen nedan.

En nod är det yttersta elementet (<Knapp ID='id'> … </Knapp>). Attributet ID som finns i det yttersta elementet används för att identifiera en nod, likt det som finns i HTMLs taggar. De inre elementen har olika roller, de tre första är till för att beskriva knappens positionering på skärmen och texten som skrivs ovan på. Det fjärde elementet är en datadriven kant som aktiverar en nod med IDet ”Home”. Inre element kan representera egenskaper hos noden eller vara någon form av exekverings drivenkant.

```
<Knapp ID='id'>
<PosX>10</PosX>
<PosY>10</PosY>
<Text>Knapp</Text>
<OnClick>Home</OnClick>
</Knapp>
```

Figur 1: Textuell programmering

4.1.3 Visuell programmeringsmetodik

På grund av att visuell programmeringsmetodik slås ut i 6.1 så bryter vi ut teorin och lägger den i bilaga 3.

4.1.4 Modulär programmeringsmetodik

Baldwin och Clark[21, s. 63] beskriver i sin bok två kärnidéer för modulär programmering:

- Moduler är enheter i stora system som är strukturellt oberoende av varandra, men som fortfarande arbetar tillsammans. Systemet som helhet måste stödja en arkitektur som tillåter självständighet i strukturen och integration av funktion.
- Andra idén fångas med tre egenskaper: abstraktion, dold information och interface. Ett komplext system kan hanteras genom att delas upp i flera mindre delar där vi ser på varje del för sig. När komplexiteten blir för stort hos ett element, isolerar vi komplexiteten genom att definiera en separat abstraktion som har ett enklare interface. Abstraktionen döljer komplexiteten i elementet och interfacet indikerar hur elementet agerar med det stora hela.

Om vi ser tillbaka på det vi tagit upp om flödesprogrammering(4.1.1) och jämför med de modulära idéerna ser vi en likhet mellan en modul och en nod då båda representerar en funktion som arbetar mot andra funktioner, men är inte starkt beroende av dessa. Vi märker ytterligare det här i Fabrik, InterCONS och Hammerwatch Editorns skriptspråk(4.1.3). Även ser vi detta drag i HTML(4.1.2) där ett element kan representera en nod. Som t.ex. knapp (<button>) eller länk (<a>). Om ett element inte är en nod så beskriver det något attribut, vilket vi finner i tabeller där celler och rader är attribut som beskriver noden.

Vi har med hjälp av Baldwin och Clarks två kärnidéer skapat oss en koppling mellan nod och modul, men ingen för kant och kommunikation. Vilket öppnar för att använda visuell pilteknik eller textuell id-teknik för att representera en kant.

4.2 Undersökning

4.2.1 Livscykel, faser och stadier

Kan läsas i bilaga 3. Läs 5.1.1 för undersökningens livscykel. Teorin riktar sig för produkter där vi utför ett antal iterationer över designen. Vi itererar endast två gånger.

4.2.2 Prototyper

Prototyper skapas för att testa idéer och koncept mot användaren. Vi tar upp två nivåer.

Low-fidelty prototyping (LOFI)[3, pp. 531]

Denna form av prototyp är lätt att skapa och ser inte ut som slutgiltig produkt. Oftast tillverkas den i andra material. Den används för att testa designidéer och koncept. Prototypen har aldrig som uppgift att behållas och integreras i slutprodukten. Den används för att

utforska funktionalitet och designval. Nedan tar vi upp några former av LOFI prototyper.

Skisser av den tilltänkta designen. Det viktiga är inte att det är vackert utan att det ger en konceptuell bild av slutprodukten. Används oftast med andra former av LOFI-prototyper.

Storyboard består av en serie av skisser som visar hur en användare kan utföra en uppgift med den tilltänkta produkten.

Prototyper med indexkort. Genom att använda indexkort som representerar olika sidor hos prototypen kan användaren ”klicka” sig fram och testa olika funktioner.

Trollkarlen av Oz antar att du har en mjukvarubaserad prototyp. Användaren sitter vid en dator och utvecklaren vid en annan. När användaren utför något svarar utvecklaren med någon form av reaktion. I princip existerar gränssnittet men reaktioner bakom sköts av en person.

High-fidelity prototyp (HIFI)[3, s. 535]

Prototypen ger en bra bild av den slutgiltiga produkten. När det kommer till applikationer är prototypen ett utvecklat gränssnitt där koden bakom returnerar konstanta värden. En HIFI Protoyp är kostsam, tidskrävande att implementera, kan testa funktionalitet, fullt interaktiv och ett klart navigationsschema. Även mer användardriven och låter användaren testa och utforska artifakten. Är en levande specifikation och används för marknadsföring och säljverktyg.

4.2.3 Scenarion

För att testa de olika prototyperna är scenariotester väldigt användbara där vi låter testpersoner sätta sig in i en situation och roll där de ska utföra någon uppgift med systemet. Detta ger även ett verktyg för att se hur användare tänker vid utförande. Som exempel kan vi i ett GPS-system be användare ställa in mål att ta sig till och sedan se hur denne tar sig fram och tänker. Scenariona kan även rangordnas i svårighetsgrader för att lägga press på designen. Information vi samlar in under insamlingsfasen används för att skapa scenarion och svårighetsgrader. En bra metod att förstå hur användare fungerar är att be dem tala högt om vad de tänker under undersökningen[3, s. 505].

4.2.4 *Datainsamling*

Under testerna är det viktigt att samla in information ifrån deltagare genom triangulering. Detta uppnås genom att ha flera olika informationsinsamlingsmetoder och någon form av datainspelning via anteckningar, anteckningar plus kamera, ljud plus kamera eller video. Ett annat verktyg för att tidigt hitta fel i en prototypdesign är pilotstudier, där vi kan fånga upp självklara fel innan de når testerna och då se till så att deltagare kan koncentrera sig på mindre självklara fel. Pilotstudiens resultat tas aldrig med i slutstudien, däremot löser den brister som funnits[3]. Brooke[5] har tagit fram en skala för att jämföra användbarheten mellan olika system som kallas System Usability Scale (SUS), som författaren beskriver som en ”quick and dirty” användbarhetsskala. Däremot påstår J. Sauro[12] att skalan är snabb och inte så smutsig som det påstås.

4.2.5 Deltagare

Eftersom vi bara kommer ha fem deltagare i undersökningen samt en testpilot är det viktigt att deltagarna är aktiva inom KÄK eller kreativa på så sätt att de antingen lajvar eller rollspelar. Vi kommer i 5.1.2 att beskriva vilka deltagare vi tog och varför.

4.3 Anveckling

A. Avdic föreslår att ordet anveckling ska användas för att beskriva fenomenet med användare som utvecklar då det blir allt vanligare. Med anveckling menar han en form av systemutveckling som initieras, drivs och/eller utförs av användare av artefakten som utvecklas[6]. En viktig definition är de roller som existerar inom anvecklingen, vilka för oss är de fyra nedanstående:

Användaren som interagerar med artefakten, det vill säga användaren av Terminalen.

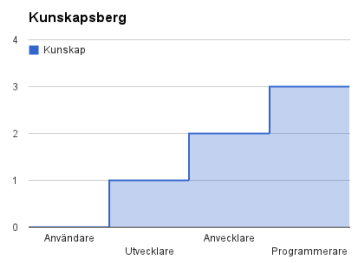
Utvecklaren som utvecklar nya terminalsidor, det vill säga användaren av utvecklingsverktyget.

Anvecklaren som implementerar ny funktion-alitet till verktyget för utvecklaren.

Programmeraren som skapar mjukvaran för användning. Tredjepartsprogrammerare är inte aktivt deltagande i utvecklingen av produkten vid skapande. De kommer in när produkten har släppts för allmänt bruk.

Något som ska betonas är att personer kan röra sig mellan rollerna och att en roll definieras av vad en person utför. Vi kan likna detta med ett berg där höjden är ”behövd kunskap” och bredden ”positioneringen jämfört med andra roller”. Vid foten hittar vi användaren, nästa platå är för utvecklaren, nästa platå efter det är för anvecklaren och vid toppen hittar vi programmeraren (Se figur 2). A. MacLean och kollegor gör en liknande jämförelse i sin artikel[9] där bredden istället är kraftfullheten hos redigering (Förmågan att ändra artifakten till användarens behov).

I tidskriftsartikeln ”Component-based technologies for end-user development” tar de upp tre tekniker för att göra det lättare för användare att gå ifrån att vara användare till anvecklare. Teknikerna är[7]:



Figur 2: Kunskapsberg

- *Direktaktivering*. Förmågan att kunna ändra på användargränssnittet och att dessa ändringar hänger sig kvar mellan körningar av applikationen. Det innebär också att redigering av funktionalitet ska kunna nås inifrån gränssnittet. Som att kunna redigera positioneringen av knappar eller andra egenskaper hos grafiska objekt.
- *Skräddarsyläge*. Både visuella och icke-visuella moduler visas. Dessa moduler kan kopplas samman genom ingångs- och utgångsportar. Du har även chansen här att redigera mindre stycken av bakomliggande kod såsom MySQL-anrop.
- *Skräddarsyklient*. Tillåter anvecklaren att modifiera i tre olika vyer. Första vyn hanterar visuella moduler, här tillåter vi att ändra storlek och positionering. Andra vyn hanterar interaktionen mellan moduler. Tredje vyn hanterar en abstrakt trädvy av ”nesting structure”.

Tanken är att anvecklaren bara behöver lära sig precis så mycket som behövs för att ändra existerande moduler och genom att använda dessa tre tekniker hoppas författarna av tidskriftsartikeln på att göra avståndet mellan användare och anvecklare mindre. Däremot kan det leda till att kunskapskravet för att anveckla sänks och då riskerar vi att begränsa mängden av möjlig utvecklingsbar funktionalitet. Det är viktigt att notera att vi inte vill minska kravet på skillnad i kunskap mellan anvecklaren och utvecklaren, bara se till att personer lättare kan förflytta sig mellan roller. En teknik som blandar skräddarsyläge och direktaktivering är att tillåta användare redigera och skapa knappar som sedan utför någon form av uppgift. Dessa uppgifter kan vara allt ifrån att följa en ordning av musmakron över olika grafiska objekt på skärmen till att sända ut mejl till olika personer. Anledningen till att det blir en blandning är att användaren kan ändra på knappens storlek och position, samt att det går att redigera den bakomliggande koden. Det krävs inte mycket inlärande för användare att utföra dessa ändringar då det finns färdiga knappar, vilket leder till att användare kan se tidigare lösningar och kopiera[9].

Gemensamt har teknikerna svagheten att det är svårt eller omöjligt att implementera ny funktionalitet. Om utvecklaren ger anvecklaren en stor mängd funktionalitet uppstår problemet att det är svårt att hitta rätt funktionalitet. Ett annat problem med anveckling är att ändringar i den uppbyggda arkitekturen kan leda till anvecklaren måste ändra eller skriva om sina implementerade moduler[8]. Problemet förekommer när en produkt utvecklas och kan lättas genom att tillåta utbyten mellan anvecklarna så att effektivare och nyare lösningar kan spridas. Om även moduler kan skräddarsys till anvecklaren upplevs en större personligkoppling och modulen blir mer en del av personen[9]. En annan teknik för att skapa lättare anveckling är att kunna gruppera moduler för att med ett knapptryck skapa avancerade lösningar. En sådan grupp skulle kunna vara ”log in” där användaren trycker och alla moduler som krävs för inloggning skapas.

Utifrån detta ser vi i att teknikerna riktar sig mot användare, utvecklare och anvecklare, vilket i kunskapsberget ute-lämnar programmeraren. Här är programmeraren en tredjepartsutvecklare då denne utvecklar produkten efter publicering. Vi kommer benämna denna form av utveckling som tredjepart. Detta innebär att vi släpper källkoden öppen, skriver ett interface eller annan abstraktion för att underlätta för tredjeparts-programmerare.

Viktigt att notera är att det sker en utveckling i arkitekturen som leder till att flera tredjeparts-programmerare oberoende av varandra kan utveckla produkten och sedan sätta in sin lösningar i samma system och få samspel.

METOD

5.1 Visuell, Textuell eller Modulär

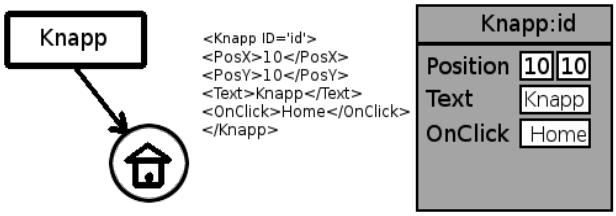
5.1.1 Livscykel

Undersökningen består av två delar; LOFI och HIFI. I första delen deltar visuell, textuell och modulär programmeringsmetodik(Se figur 3). När första delen är klar passerar två vidare. Efter sista ska endast en metodik finnas kvar. Varje steg består av tre delar; implementation, pilotstudie och undersökning. I varje del deltog sex personer där fem var med i undersökningen och en i pilotstudien. Deltagarna var kreativa personer (På fritiden utövade de kreativa aktiviteter så som lajv och rollspel.) som hade ingen eller liten erfarenhet inom gränssnitts-utveckling. Nedan följer en kort beskrivning av varje del. Vi kommer gå in djupare på dessa i 5.1.3 och 5.1.4.

Implementation. Här skapade vi prototypen och scenarion. Mycket tid går till efterforskning över hur de olika programmeringsmetodikerna kan implementeras och scenariorna bygger på vad KÄK vill att användare ska utföra med Terminalen som att logga in och sända mejl.

I *Pilotstudien* använder vi prototypen och scenariona. Resultaten tas inte med i slutrapporten. Däremot rättar vi till fel som upptäcktes (Stavning, logik eller liknande). Pilotstudien genomförs som en semi-strukturerad intervju riktad mot icke-strukturerad[3, p. 299], där vi diskuterar designval, utseende och språk. Deltagaren uppmuntras att inte bespara designen några känslor, helt enkelt berätta vad deltagaren tycker och inte vara rädd för att ge återkoppling.

Undersökningen är uppdelad i tre delar; presentation, inriktad och övergripande.



Figur 3: Visuell, textuell och modulär programmering

I presentationsdelen började vi med att fråga deltagaren om tidigare erfarenheter inom programmering. Om den saknades gick vi kort igenom konceptet. Efteråt presenterades prototypen och dess syfte.

I den inriktade delen började vi med att presentera en av utvecklingsmetoderna. Deltagarna fick utföra scenarion(se 5.1.3 och 5.1.4) och svara på frågor rörande programmeringsmetodiken. Detta steg upprepades tills alla metodiker var utvärderade. Frågor som användes skiljde mellan LOFI och HIFI.

I övergripande delen fick deltagaren svara på frågorna;

- Nämn något positivt och negativt om varje språk.
- Ranka språken efter vilken du helst vill använda.

Här fick de chansen att ändra sig rörande tidigare frågor. Språk syftar då på programmeringsmetodiken.

Undersökningen skedde anonymt på så sätt att deltagaren aldrig behöver uppge namn och vi valde att inte spela in video eller ljud. Undersökningen genomfördes som en semistrukturerad intervju[3, p. 299]. Informationen samlades in via de ställda frågorna och observationer vid användandet av prototypen. Vi uppmuntrade deltagarna att vara öppna och berätta vad de tyckte.

Resultaten ifrån undersökningen användes för att avgöra vilken metodik som gick vidare.

5.1.2 Deltagare

Testpiloten som deltog tillhör KÄK och valdes dels för dennes kreativa sida med högt engagemang i både lajv och rollspel dels för bristande kunskaper inom programmering och datoranvändning. Samma testpilot användes för LOFI och HIFI. De fem andra deltagarna som valdes ut var:

En deltagare som har erfarenheter inom programmering, men inte är färdigutbildad i området. Har hobbykodat med olika språk eller läst någon kurs på högskolan som rör området. Deltagaren representerar gruppen inom KÄK med djupare kunskap inom mjukvaruutveckling. Denna deltagare var med i både LOFI- och HIFI-undersökningen för att ge en yttre erfarenhetssyn på Verktyget.

Två deltagare som har erfarenheter inom lättare programmering, så som webdesign, speciellt HTML. Däremot har de ingen vidare drivkraft för att lära sig mer om området. De representerar steget mellan deltagaren vi nämner ovan och de vi nämner nedan.

Två deltagare utan erfarenheter inom programmering. Verktyget kommer ge störst utmaning för dessa användare. De var olika mellan LOFI och HIFI, för att deltagaren efter en undersökning har fått för mycket kunskap om verktyget och området.

5.1.3 *LOFI prototyp*

Du kan läsa om vad som skiljde mellan 5.1.1 och LOFI-prototypen under ”Prototyp” i bilaga 1.

5.1.4 *HIFI prototyp*

Du kan läsa om vad som skiljde mellan 5.1.1 och HIFI-prototypen under ”Prototyp” i bilaga 2.

5.2 Anveckling

Eftersom skräddarsyklient är det vi kommer fram till (6.3) ska användas, kan du läsa allt om direktaktivering, skräddarsyläge och tredjepart i bilaga 4. Även hur dessa svarar på frågorna vi kommer fram till i 5.2.2.

5.2.1 Tekniker

Skräddarsyklient

Klienten är ett yttre program som redigerar objekt för verktyget och terminalen. Varje objekt består av mindre komponenter som implementerar en specifik funktionalitet.

Visuell. Anvecklaren ser varje objekt i sin grafiska representation ifrån Verktygets syn. Genom att välja ett objekt kan anvecklaren se underliggande komponenter. Varje komponent kan ha inre och yttre in- och utportar. Inre hanterar kommunikation mellan komponenterna och representeras av pilar. Yttre tar hand om kommunikation mellan objekt och representeras med pilar – denna form kan inte användas i skräddarsyklienten.

Textuell. Anvecklaren ser varje objekt som samlingar av element som omges av ett element med namnet på objektet. Inre elementen är komponenter. Genom att välja ett objekt kan anvecklaren redigera de inre elementen som representeras i text. Kommunikation mellan objekt skötts via IDn och det finns element som representerar ingångs- eller utgångsportar. Inre kommunikation är inte redigerbart utan ett element ställer krav på objektet att andra element måste finnas.

Modulär. Anvecklaren ser varje objekt som ett modul-block innehållandes flera komponenter. Kommunikationen fungerar likt den visuella, men varje komponent har en privat ID istället för pilar.

5.2.2 Frågor

Från vad vi ser i anvecklingsteorin (4.3) har vi tagit fram dessa frågor som rör vad vi anser är viktigt för anveckling att svara på.

- Vilken utvecklingsmiljö sker anvecklingen i?
- Hur brant är inlärningskurvan?
- Hur lägger anvecklaren till nya moduler?
- Hur lägger anvecklaren till ny funktionalitet?
- Hur ändrar anvecklaren på redan existerande moduler?
- Hur tar anvecklaren bort moduler?
- Hur sprider anvecklaren skapade moduler?

5.2.3 Skräddarsyläge

Se 5.2.

5.2.4 Skräddarsyklient
Vilken utvecklingsmiljö sker anvecklingen i? En applikation skapad av utvecklarna specifikt för att utveckla moduler till Terminalen och Verktyget.

Hur brant är inlärningskurvan? Medelbrant. Mycket av det arbete som sker är att skapa moduler som består av komponenter, vilket kan jämföras med att bygga med legobitar.

Hur lägger anvecklaren till nya moduler? Anvecklaren skapar en ny modul i verktyget som sedan fylls med komponenter. Sparas till hårdisken så att Terminal och Verktyg kan läsa in filen.

Hur lägger anvecklaren till ny funktionalitet? Det går inte att lägga in ny funktionalitet, men det går att skapa en illusion av detta genom att sätta ihop olika komponenter. Mängden är begränsad genom att det bara existerar ett visst antal kombinationer av komponenter.

Hur ändrar anvecklaren på redan existerande moduler? Anvecklaren väljer en objektfil ifrån hårdisken och laddar in. Användaren kan då lägga till eller ta bort komponenter. Ser även till så att inre kommunikation sker korrekt mellan komponenterna.

Hur tar anvecklaren bort moduler? Anvecklaren raderar en modul via att ta bort objekt-filen på hårddisken.

Hur sprider anvecklaren skapade moduler? Anvecklaren kan direkt sprida enskilda moduler. Däremot måste mottagare göra ett val mellan existerande modulen och den nya om det redan finns en.

5.2.5 Tredjepart
Se 5.2.

RESULTAT

6.1 LOFI prototyp
Se Bilaga 1. För pappersprototyp och scenarioresultat. Uträkning av Tabell 1s värden förklaras också i bilagan.

Implementation

Vi skapade en pappersprototyp med tre scenarion.

Pilotstudie

Textuell hade bristande dokumentation vilket lede till svårigheter för piloten.

Visuell hade bristande bilder vilket lede till stor förvirring hos piloten. Bilderna blev självklara vid förklaring.

/	Överblick	Kontroll	Naturligt	Rank
Modulär	3,7	3,6	3,3	11
Visuell	3	2,8	2,4	7
Textuell	4	3,8	3,8	12

Tabell 1: Resultat ifrån LOFI undersökningen

Modulär fick minst av både positiv och negativ återkoppling.

Allmänt. Det är viktigt i vilken ordning programmerings-metodikerna presenteras för deltagaren. Tre scenarion tar för lång tid och tar deltagarens energi.

Ändringar

Textuell. Skriven dokumentation för hur taggar ska formuleras och vad för information som finnas inom varje tagg.

Visuell. Vi tog bort noder som inte används i första scenariot.

Modulär. Vi la in fler av redan existerande moduler för att ge mer arbetsyta till deltagaren: Två text-, en knapp- och en fältmodul.

Allmänt. Vi ändrade alla texter till Engelska (Existerade knappar och moduler som hade svenska namn). Viktigaste ändringen är att istället för att använda tre scenarion använder vi nu bara det första.

Undersökning

Textuell fick mest kritik gällande komplexitet och dokumentation. Den fick även minst positiv återkoppling. Deltagare kände att de visste exakt vad de får. Det var lätt att spåra tidigare inlägg och gav bra översikt.

Visuell fick skarpast kritik, mycket rörande tydlighet och tolkning av bilder. Positiv återkoppling var att det gav en känsla av ökad kontroll.

Modulär upplevdes som snabbt att använda och som en verktygslåda. Deltagare tyckte det var jobbigt att använda musen och positionering med koordinater. De upplevde att det kan bli rörigt om det blir många moduer samtidigt.

Allmänt. Det är viktigt att deltagaren får respons på agerande i miljö.

Medelvärdet ifrån frågorna, som nämnts i 5.1, kan ses i Tabell 1. Rank är det sammanslagna värdet ifrån sorteringen. Den metodik som deltagaren ville använda mest fick tre poäng och minsta fick ett.

Forstättning

Vi tog bort den visuella programmeringsmetodiken.

6.2 HIFI prototyp
Se Bilaga 2. För HIFI prototyp och scenarioresultat.

Implementation

Positioneringen är nu centrerad istället för att utgå från övre vänstra hörnet. Krav ställt ifrån KÄK.

Textuell dokumentation sker genom att trycka på en knapp med modulens namn och ett Pop-up fönster visas med taggsyntaxen.

Pilotstudie

Textuell. Piloten upplevde representationen av den inbyggda dokumentationen mer som ett hinder en hjälp.

Utan syntaxmarkering på texten var det svårt att avgöra vad som är vad.

Modulärt. Piloten upplevde att det var lätt. Däremot svårt att koppla ihop hur IDn representeras och används, vilket skapade stor förvirring.

Allmänt. Piloten hade problem med avsaknad av uppvärmning/startperiod, det blev en för brant inlärningskurva att köra ett tyngre scenario ifrån start. Tooltip på de olika knapparna var användbart. Vi körde direkt scenario två.

Ändringar

Textuell. Vi implementerade så att dokumentationen kunde visas samtidigt som koden skrevs samt syntaxmarkering och radnummer för koden.

Modulärt. Vi designade om hur IDn visades ifrån "NAMN id" till "NAMN (ID: id)" där NAMN är namnet på modulen.

Fördjupar uppvärmningsscenariot till ”Skapa en sida med en knapp som visar ett Pop-up meddelande och en text som säger 'välkommen'. Introduktionsscenario.” för bättre uppvärmning.

Undersökning

Textuell. Deltagarna upplevde en känsla av full kontroll, men att det skulle vara svårare att utföra för nybörjare. Den inbyggda dokumentationen upplevdes användbar.

Modulär. Deltagarna upplevde en känsla av klarhet. Det pekades ut att vid större mängder av moduler skulle det bli svårt att se vad som gjorde vad och att det vore bra om vi kunde ändra IDn själva.

Allmänt. Uppvärmningsscenario innan det svåra och sedan återkoppling mot det gjorda arbetet fick deltagare att reagera positivt. Något som deltagare reagerade negativt på var bristen återkoppling som uppstod när de skrev in IDn. De visste inte vart IDn ledde. Det var svårare för deltagarna att kritisera prototypen då den såg välgjord ut.

I Tabell 2 kan vi se sammanfattade resultaten ifrån undersökningen. Hur detta räknades ut förklaras i Bilaga 2 och hela resultatet finns där.

6.3 Anveckling

6.3.1 Anvecklingsverktyg

Skräddarsyklient är steget mellan skräddarsyläge och tredjepart. Det är varken för enkelt eller avancerat. Vi utövar redan en form av redigering i Verktyget. Därför ska KÄK använda Skräddarsyklient till Verktyget. När vi benämner skräddarsyklienten i bestämd form syftar vi på den applikation som KÄK ska använda.

Mer om hur skräddarsyklienten ser ut hittar du i bilaga 5.

	SUS[5]	Rank
Modulär	76	6
Textuell	69	8

Tabell 2: Resultat ifrån HIFI undersökningen

6.3.2 Arkitektur – Systemdelar

Ett system är en implementation av arkitekturen. Inre systemdelar är delar som är implementerade på samma kärna. Vad en kärna är tar vi upp i Kärnarkitektur.

Förklaring till Figur 4: Blått är sparfil likt en HTML-fil. Grönt är en applikation. Orange är sparade objekt som baserar sig på en klass. Gult är klasser som ärver en basklass. Pilarna representerar relationer mellan olika systemdelar.

Stängda pilar betyder inre systemberoenden, vilket betyder att hela systemet måste basera sig på samma kärna. D.v.s. att alla tre applikationer måste använda samma Modul- och Komponentklass. Pilarnas riktning anger beroende och kontrollflöde.

Öppna pilar betyder yttre systemberoenden, vilket betyder att systemdelen kan komma ifrån ett yttre system, men fortfarande tolkas av det inre systemet. Pilarnas riktning anger dataflöde.

Relationerna behandlar vi i 6.3.3. Du kan hitta mer om varje systemdel i bilaga 5.

6.3.3 Arkitektur – Relationer

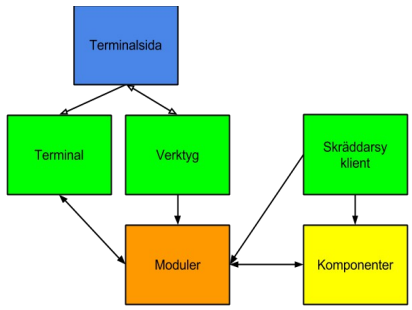
I Figur 4 kan vi se en arkitekturöversikt. Relationerna i Figur 4 beskriver vi kort nedan. För en djupare förklaring se bilaga 5. Värt att notera att på grund av Terminalsida är en sparfil så blir relationerna mot systemdelen algoritmer istället för APIer.

Verktyg ↔ Terminalsida. Verktyg kan skapa och redigera Terminalsida.

Terminalsida → Terminal. Terminal läser in Terminalsida. Kan liknas med HTML-fil och en webläsare.

Terminal ↔ Moduler. En yttre tagg i Terminalsida kopplas till en modul. Relationen används för att översätta text till objekt.

Verktyg → Moduler. Används för att bygga Terminalsida. Verktyget tolkar en modul och modulen ger vilken/vilka taggar som ska användas.



Figur 4: Arkitekturöversikt

Skräddarsyklient → Moduler. För att redigera Moduler.

Skräddarsyklient → Komponenter. För att kunna sätta in Komponenter i en modul.

Moduler ↔ Komponenter. För att Moduler och Komponenter ska kunna leva i symbios.

DISKUSSION

7.1 Metod

7.1.1 Visuell, Textuell eller Modulär

Metoden svarar på frågeställningen "Vilket av visuell, textuell och modulär programmeringsmetodik lämpar sig bäst för grafisk gränssnittsutveckling?". Däremot finns det brister som ändrar på resultatet och om de blir lösta leder till både högre validitet och reliabilitet. Anledning till att de aldrig blev lösta under metodens gång var för att de upptäcktes först efter genomförande.

De inriktade frågorna(5.13 och 5.1.4) i LOFI och HIFI skiljer sig åt vilket gör det svårt att jämföra undersökningarna. Detta resulterar i att det är svårt att se hur utvecklingen gått för prototyperna mellan de olika delarna. Däremot kan vi fortfarande jämföra de olika metodikerna inom varje del. Dock så brister LOFI för att frågorna är inte tillräcklig triangulerande eller beprövade. För att lösa detta skulle SUS ha använts igenom hela undersökningen. Anledning till att SUS bara användes i HIFI var att det hittades efter att LOFI var genomförd. Hade vi haft SUS i HIFI och LOFI hade det löst problemet.

Antalet iterationer var bara en i både LOFI och HIFI. Vilket skapar problemet att vi mäter mer brister i designen hos varje prototyp och hur detta ska lösas istället för hur väl de passar till gränssnittsutveckling. Detta hade gått att lösa genom att först iterera en gång för att lösa designproblem och sedan göra en större undersökning där vi koncentrerar oss på att jämföra de olika metodikerna. Pilotstudien användes för att lösa detta problem, men visade sig inte vara tillräcklig. Det hade behövts en till pilot.

Antalet deltagare vid varje steg var för litet vilket resulterar i att vi finner den metodik som passar för dessa deltagare och inte den stora massan. Detta löstes genom att vi valde ut specifika deltagare. För att lösa problemet bättre i framtiden ska vi ha med fler deltagre för att bygga en breddare bas.

Datainsamlingsmetoden som användes kunde förbättras, om vi istället spelat in deltagarnas reaktioner och kommentarer hade det varit lättare att dra slutsatser även långt efter undersökningen. Däremot förblir undersökningen inte anonym om vi lämnar ut inspelad data. Detta kan lösas genom att skriva ner konversationer. Däremot blir det svårare med video då detaljer kan bli försummade.

Metoden är replikerbar och prototyperna finnas i Bilaga 1 och 2. Däremot kommer resultaten att skilja beroende på vilken målgrupp av deltagare som används. Vi måste även tänka på att människans mångsidighet gör såna här mätningar svåra.

Med metoden finner vi vilken programmeringsmetodik som lämpar sig bäst för gränssnitssutveckling, men inte lika precist och exakt som vi hoppats på. Till slut hittade vi den metodik som deltagarna ansåg lämligast för utveckling, vilket vi tar upp i 7.2.1.

Avslutningsvis så var metoden väldigt lovande innan undersökning, däremot var bristerna uppenbara efter genomförande. I framtiden måste vi bemöta och lösa bristerna ovan.

Källkritik

De källor som användes i förstudien var inte tänkta för att användas på så sätt att vi utvecklar flera produkter samtidigt och jämför dessa med varandra. Istället riktade den sig mot att utveckla en produkt och göra den mer och mer användarvänlig. Mycket av metoden togs ifrån Rogers, Yvonne, Helen Sharp, och Jenny Preeces bok ”Interaction design: beyond human-computer interaction”[3].

7.1.2 Anveckling

Metoden som används tar fram tre tekniker för anveckling, men går aldrig djupare in på arkitekturen bakom dessa utan riktar sig mest åt anvecklingen och den slutgiltiga arkitekturen, vilket betyder att vi närmar oss frågan ifrån anvecklingsperspektivet istället för det arkitekturella perspektivet. Metoden fungerar för att hitta en lösning, men hade följande brister:

- Ingen matematisk metod för att jämföra olika tekniker. Vilket betyder att vi har fått förlita oss på vår slutledningsmetoden att om vi har A, B, C och D är på B och vill förflytta oss till nästa punkt hamnar vi på C.
- Arkitekturen har aldrig implementeras eller testats. Vilket betyder att arkitekturen kommer med ett antal barnsjukdomar som är svåra att förutspå innan implementation.
- Endast tre tekniker tas upp i metodkapitlet och de är alla metoder för anveckling. Dock riktar de sig mot att anveckla olika delar av systemet och att förflytta sig mellan användare och anvecklare.

För att lösa bristerna hade vi behövt mer tid för att implementera och hitta en metod för att jämföra olika tekniker. Jämförelse metoden kan vara en SUS-undersökning för varje teknik riktad mot KÄK och utveckling av Terminalen. Vi tog fram en arkitektur för anveckling i ett modulärt system. Därefter måste arkitekturen implementeras och testas. Som avslutning vill vi säga att metoden saknar en mätbar teknik för att jämföra olika tekniker och vi får istället använda en slutledningsmetod för att finna resultatet.

Källkritik

Litteraturen som används i arbetet riktade sig mot att minska avståndet mellan användare, anvecklare och

utvecklare med den metoden att vi gör övergången lättare vilket gör att vi får de tre stegen (direktaktivering, skräddarsyläge och skräddarsyklient). Vi avgränsade oss på så sätt att vi såg varje steg som en teknik för anveckling istället för en hel lösning där varje steg underlättar förflyttning.

7.2 Resultat

7.2.1 Visuell, Textuell eller Modulär

LOFI

Utifrån att världen vi lever i upplevs visuellt och inte textuellt/modulärt utgick vi från att visuell programmering skulle vara accepterat av användare. Vi antog att personer skulle rangordna metodikerna; visuell, modulär och textuell där visuell är den mest föredragna. Något vi efter undersökningen fick se som fel och att personer tolkat bilder olika och att ord betyder lika för alla. Om visuell programmering testas igen måste vi med någon metod ge klarare bilder och skapa en standard som alla förstår utan förvirring. En annan lösning kan vara att byta bilder mot text vilket skulle skapa klarare visuell programmering.

HIFI

Utifrån SUS skala och att den riktar sig mot system som ska bli eller vara användarvänliga för den generella massan antog vi att verktyg för att skapa gränssnitt skulle få ett lägre värde än 60.0 och då hamna under betyg C. Detta visade sig felaktigt då de båda fick betyg B där modulär uppnådde bäst resultat.

Även om den modulära programmeringsmetodiken fick bäst värde på SUS skalan så ville personer helst utveckla med den textuella, vilket vi ser genom rankningen där modulär rankades 6 och textuell 8. Det skapar ett problem som måste analyseras. Deltagare gav minst positiv och negativ återkoppling om modulärt i båda undersökningarna. De hade mest kritik mot den textuella, men valde fortfarande den textuella över modulära. Detta grundar sig troligtvis i tryggheten vi finner i text tack vare det vardagliga bruket.

Slutresultat

Från HIFI-resultatet ska KÄK använda modulär programmeringsmetodik då deltagarna överlag upplevde metodiken som lättare och gav den högre SUS värde. Även om textuella fick högre rankning så pekar undersökningen på modulärt.

Om vi använder den arkitekturlösningen som föreslås i 6.3 kan vi implementera ett system där både textuella och modulär programmeringsmetodik kan användas. Däremot skulle Verktyget endast stödja modulär metodik och den textuell metodiken skulle bara återfinnas i terminalsidornas filer.

7.2.2 Anveckling

Resultatet vi nämner i 6.3 behöver inte vidare analyseras och inga ytterligare kommentarer finns för resultatet.

7.3 Arbetet i ett vidare sammanhang

7.3.1 Visuell, Textuell eller Modulär

En klar lärdom är att det är viktigt att vara positiv och göra undersökningarna mer till en lek. Att öppna med försiktig kritik mot det egna arbetet för att släppa spänningar som existerar mellan deltagare och undersökare. Har vi få deltagare är det viktigt att de kommer från olika målgrupper som alla tillhör huvudmålgruppen.

7.3.2 Anveckling

Från detta arbete skapar vi en teknik för anveckling mot utvecklingsverktyg och med den framtagna arkitekturen finns det något att implementera och köra tester mot, vilket vi inte har stött på i annan litteratur under arbetetsgång.

SLUTSATSER

Syftet uppnåddes med arbetet. Vi hittade en lösning på båda frågeställningarna.

Vilket av visuell, textuell och modulär programmeringsmetodik lämpar sig bäst för grafisk gränssnittsutveckling?

Modulär programmeringsmetodik fick bästa resultat i undersökningarna. Däremot kom textuell programmeringsmetodik så pass nära att det är ett substitut som kan användas. I KÄKs fall så ska de använda den modulära metodiken. Med detta så kommer KÄK veta vilket metodik som ska användas i Verktyg när det implementeras och kommer i framtiden få det lättare att utveckla Terminalen.

Hur kan en modulär arkitektur realiseras så att systemet medger hög utvecklingsbarhet?

För den arkitektoniska lösningen se 6.3. Med denna lösning har KÄK nu en ”ritning” för vad som behöver göras för att lösa sina behov med utveckling. När lösningen är implementrad kommer KÄK kunna skapa moduler med Skräddarsyklienten som kan användas i både Terminalen och Verktyget. De kan även sprida lösningar de gjort mellan sig genom att sända modulfiler. Modulerna tolkar också bara terminalsidor. Dessa egenskaper leder också till att om de har flera Terminaler kan de göra varje Terminal personlig genom att ha olika modulfiler på varje dator.

Framtida studier

Implementera en Visuell HIFI prototyp. Få bort designfel för alla tre prototyper genom metoden som tas upp i 4.1. Utför en ny undersökning med fler deltagare. Blir resultatet annorlunda?

Implementera arkitekturen vi har tagit fram för anveckling. Vilka barnsjukdomar finns och hur löser vi dessa? Hur implementeras arkitekturen i Java eller finns det ett bättre språk? Vilket SUS-värde får tekniken i en undersökning?

REFERENSER

[1] Gould, J. "How to design usable systems (excerpt)." *IBM Research Center Howthorne Yorktown Heights, New York* 10598 (2000): 757-789.

[2] Mayhew, Deborah J. "The usability engineering lifecycle." *CHI'99 Extended Abstracts on Human Factors in Computing Systems*. ACM, 1999.

[3] Rogers, Yvonne, Helen Sharp, and Jenny Preece. *Interaction design: beyond human-computer interaction*. John Wiley & Sons, 2007.

[4] Ferré, Xavier, et al. "Usability basics for software developers." *IEEE software*18.1 (2001): 22-29.

[5] Brooke, John. "SUS-A quick and dirty usability scale." *Usability evaluation in industry* 189 (1996): 194.

[6] Avdic, Anders. "Användare och utvecklare: om anveckling med kalkylprogram. "*MTOAnvändarperspektivet* (2001): 105.

[7] Mørch, Anders I., et al. "Component-based technologies for end-user development." Communications of the ACM 47.9 (2004): 59-62.

[8] Fischer, Gerhard, and Andreas Girgensohn. "End-user modifiability in design environments." Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 1990.

[9] MacLean, Allan, et al. "User-tailorable systems: pressing the issues with buttons." Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, 1990.

[10] Nielsen, Jakob. "The usability engineering life cycle. " *Computer* 25.3 (1992): 12-22.

[11] Gould, John D., and Clayton Lewis. "Designing for usability: key principles and what designers think." *Communications of the ACM* 28.3 (1985): 300-311.

[12] Sauro, Jeff. "Measuring usability with the system usability scale (SUS)." (2011).

[13] Nielsen, Jakob. "Usability Engineering." (1993).

[14] Hammerwatch Editor. [ONLINE] 2014-05-30 http://www.hammerwatch.com/

[15] Raggett, Dave, Arnaud Le Hors, and Ian Jacobs. "HTML 4.01 Specification."*W3C recommendation* 24 (1999).

[16] Bray, Tim, et al. "Extensible markup language (XML)." *World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210* (1998).

[17] dom4j [ONLINE] 2014-05-30 http://dom4j.sourceforge.net/

[18] Hils, Daniel D. "Visual languages and computing survey: Data flow visual programming languages." *Journal of Visual Languages & Computing* 3.1 (1992): 69-101.

[19] Ingalls, Dan, et al. "Fabrik: a visual programming environment." *ACM SIGPLAN Notices*. Vol. 23. No. 11. ACM, 1988.

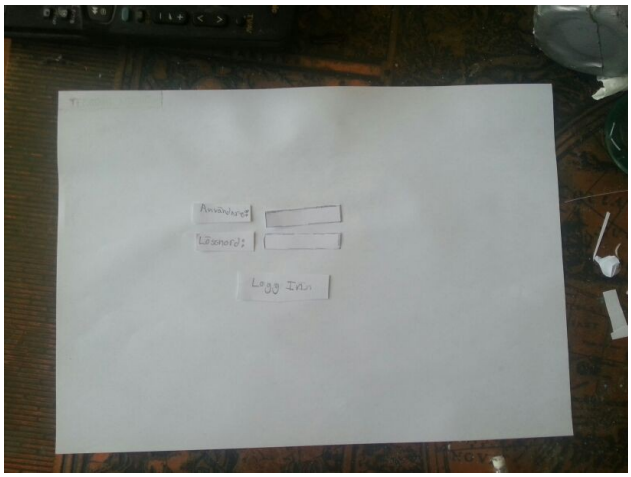
[20] Smith, David N. "Visual programming in the interface construction." *Visual Languages, 1988., IEEE Workshop on*. IEEE, 1988.

[21] Baldwin, Carliss Young, and Kim B. Clark. *Design rules: The power of modularity.* Vol. 1. Mit Press, 2000.

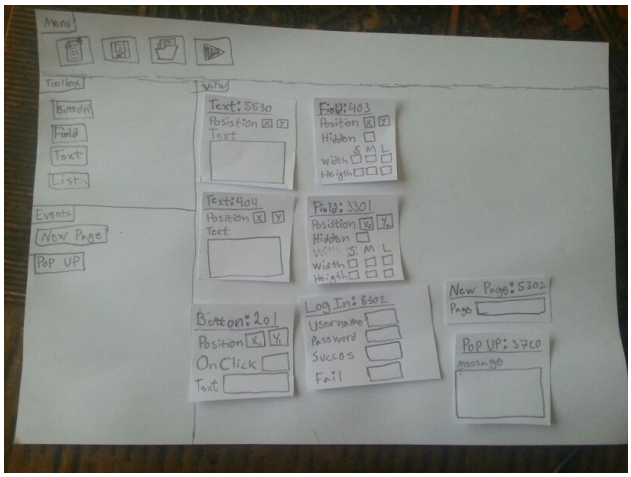
[22] Wrap Layout [ONLINE] 2014-06-01 http://tips4java.wordpress.com/2008/11/06/wrap-layout/

[23] XML Text Editor [ONLINE] 2014-06-01 https://code.google.com/p/xml-text-editor/

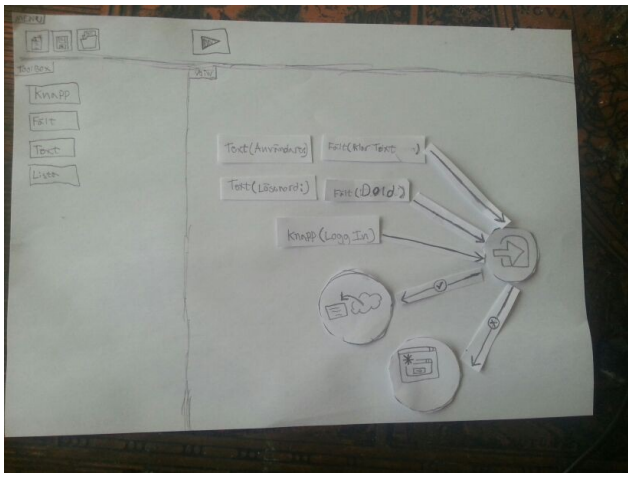
[24] Text Component Line Number [ONLINE] 2014-06-01 http://tips4java.wordpress.com/2009/05/23/text-component-line-number/



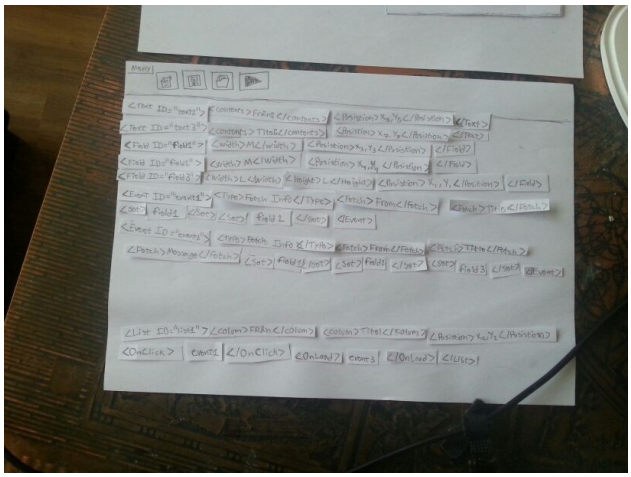
Figur 1: Mål för användaren vid första scenariot



Figur 2: LOFI(Modulär) mål för användaren vid första scenariot



Figur 3: LOFI(Visuell) mål för användaren vid första scenariot



Figur 4: LOFI(Textuell) mål för användaren vid första scenariot

LOFI Prototyp

BILAGA 1: PAPPERSPROTOTYP OCH RESULTAT

Prototyp

Figurerna ovan.

Implementation

Vi började med efterforskning om hur visuell, textuell och modulär programmeringmetodik kan se ut och fungera. Vi tog fram tre olika scenarion:

1. Deltagaren skulle skapa en "log in" sida.
2. Deltagaren skulle lägga till texten "Från:" på en "skicka mejl" sida.
3. Deltagaren skulle skapa en sida för att visa inkorg och läsa mejl.

Scenarierna hölls abstrakta för att ge bättre inblick i vad en användare skulle tänkas behöva för att uppnå slutmålet. När scenariorna var framtagna tillverkades en pappersprototyp. Den var uppdelad i två delar; en resultatdel som visade vad deltagaren åstadkommit genom utvecklarverktyget och själva utvecklargränssnittet där deltagaren skrev koden.

Pilotstudie

Utöver det som nämnts tidigare (5.1.1) koncentrerade vi oss ytterligare på avsaknaden av funktionalitet (Saknades någon form av knapp, text eller grafiskt objekt) och uppdelningen av scenarion (Är de för många eller stora? Vad är oklart?).

Undersökning

I presentationsdelen uppmanades deltagare att lägga till egna knappar eller funktioner.

Frågorna som används på den inriktade delen är

- Känner du att du får en bra överblick?
- Känner du att du har kontroll?
- Känns det naturligt? Är det så här du vill arbeta.

Dessa frågor svarade deltagaren på efter varje utvärderad metodik med en siffra ifrån ett till fem där fem är mycket bra och ett är mycket dåligt.

Metodikerna presenterades i ordningen modulär, visuell och sist textuell. Modulär testades först för att den var bra presentation för programmering och verktyget. Textuell sist för att den tog längst tid att testa.

\	Överblick	Kontroll	Naturligt	Rank
Modulär	3,7	3,6	3,3	11
Visuel	3	2,8	2,4	7
Textuel	4	3,8	3,8	12

Tabell 1: Sammanfattade resultat från Tabell 2.

Resultat

Värdena i Överblick, Kontroll och Naturlig räknades utgenom att för respektive fråga och metodik slå ihop alla värden och dela på helheten. Värdena kommer ifrån Tabell 2.

Modulär Överblick: (3.5 + 4 + 4 + 3 + 4) / 5 = 3.7

Rank räknades ut genom 1, 2 och 3 raderna i Tabell 2. Ligger ett språk på första plats fick de tre poäng och ett poäng om de låg på sista. 1, 2 och 3 representerar då ranknings frågan.

Visuel: 1 + 1 + 1 + 1 + 3 = 7.

NR	Ö	K	N	1	2	3	Bra	Dåligt
1	3,5/2/5	4/2/5	3.5/3/4	T	M	V	**Visuell:** Bra med pilar. **Textuell:** Bra skit. Man vet exakt vad man får. **Modulär:** Snabbare att skriva, Behöver inte göra allt ifrån grunden, Lätt att hantera. Modulärt är bättre på alla sätt än visuelt.	**Visuell:** Otydligt, ingen kontroll, passar inte. För handikappande. **Textuell:** Man måste vara smartare. Tar längre tid att skriva. När man kan går det snabbt. Sämre för nybörjare bättre för erfarna. **Modulär:** Fortfarande måste man använda musen.
2	4/2/5	3/2/4	2/1/5	T	M	V	**Visuell:** Inget. **Textuell:** Full kontroll och vet man vad man gör kan man fixa allt. **Modulär:** Man fick en idé vad det olika sakerna gjorde eftersom det var text på det man satte ut.	**Visuell:** Man hade inte någon aning om vad någonting gjorde. Var dåligt dokumenterat. Obegripligt. **Textuell:** Kan ta lite tid att knacka kod. **Modulär:** Vet inte.
3	4/2/3	5/3/4	4/1/3	M	T	V	**Visuell:** Slapp positionering. Ökar kontrollen lite. Borde finnas mer tooltips! **Textuell:** Bra översikt hela tiden. Lätt att backtracka vad man har gjort. Lätt att editera om man har en manual. **Modulär:** Känns som mycket arbete redan var gjort. Känndes som en verktygslåda. Simidigt.	**Visuell:** Gillar den icke. Svår att förstå. Svår att hantera. **Textuell:** Långsam. Komplex. **Modulär:** Jobbigt med x och y positioner.
4	3/4/4	3/3/4	4/4/4	T	M	V	**Visuell:** Den är simpel när man lärt sig den. **Textuell:** Dokumentation! Man får en bra överblick. **Modulär:** Väldigt ren. Ser inte ut som ett fågelbo. Lätt lärt.	**Visuell:** Väldigt förvirrande att tolka bilderna. **Textuell:** Måste hålla kol på start och avslut annars fungerar det inte. Svåraste i början att lära sig. **Modulär:** Om man inte kan positionera rätt blir det svårt.
5	4/5/3	3/4/2	3/4/3	V	M	T	**Visuell:** Passar min still. Inte så mycket koda in tänk. Gör man det bra är det lättare att strukturera upp det. **Textuell:** Har man lite erfaranhet så kan vi se att det fungerar. Det känns onödigt krånlig. Känns onödigt krånligt på papper. **Modulär:** Inget direkt att säga.	**Visuell:** Väldigt lätt blir rörigt. **Textuell:** Väldigt krånligt om man inte är välinstatt. Svårt att få en bild av hur det ser ut. **Modulär:** Behöver man bygga ut det lite större kan vi se att det är svårt att se vad som hör till vad.

Tabell 2: Resultat ifrån LOFI undersökningen

Ö = Känner du att du får bra överblick? **K** = Känner du att du har kontroll? **N** = Känns det naturligt? **T** = Textuell **M** = Modulär **V** = Visuell **Bra** = Vad var bra med de olika metodikerna. **Dåligt** = Vad var dåligt med de olika metodikerna. **X/Y/Z** = X,Y,Z representerar given poäng för metodik i frågan. **X** = Modulär, **Y** = Visuell, **Z** = Textuell **1, 2 och 3:** Representerar rankningen(1 deltagaren ville fortsätta använda språket, 2 varken eller och 3 ville inte använda språket igen).

BILAGA 2: HIFIPROTOTYP OCH RESULTAT

Prototyp

Kan laddas ner från:

https://www.dropbox.com/s/mnli0vvdtp4jrkq/Project.rar

Visuell programmeringsmetodik hanteras ej i HIFI-testerna för att metodiken fick sämst resultat i LOFI-testerna (6.1). Vi implementerar då endast modulär och textuell programmeringsmetodik i HIFI prototypen.

Implementation

Prototypen utvecklas för att testas på dator med metodikerna som gick vidare ifrån LOFI. Dessa implementeras i samma prototyp. Via ett knapptryck kan deltagaren växla mellan modulär och textuell programmeringsmetodik. Vi kommer först utvärdera en metodik åt gången för att i slutet av undersökningen låta deltagaren växla mellan metodikerna för att lättare kunna svara på de övergrip-ande frågorna (5.1.1).

Två nya scenarierna skapades för att utförligare testa de två kvarstående metodikerna.

1. Skapa en välkomstsida. Scenariot används för att deltagaren ska få chansen att bekanta sig med verktyget.
2. Skapa en "Log in"-sida som tar deltagaren till välkomstsidan. Misslyckas deltagaren att logga in ska ett felmeddelande visas.

Vi utvecklar HIFI prototypen med:

- Java
- Eclipse EE
- Java Swing
- Dom4j[17]
- XML Text Editor[23]
- Wrap Layout[22]
- Text Component Line Number[24]

Pilotstudie

Pilotstudien koncentrerar sig ytterligare på navigationssvårigheter och problem med designen. Vi disskuterar även skillnader mellan metodikerna och tankar om dessa.

Undersökning

I presentationsdelen går vi även igenom prototypen, hur den fungerar som verktyg. Om deltagaren trycker på kör öppnas ett fönster med det innehåll de har skapat och liknande.

I den inriktade delen använder vi skalan *System Usability Scale (SUS)*.

Metodikerna presenteras i ordningen modulär och sist textuell. Med samma anledning som vi nämner i LOFI.

Anveckling

För att realisera en modulär arkitektur tar vi upp tre olika tekniker och presenterar hur de löser sex viktiga frågor inom anveckling. På grund av artefaktens form kommer vi att behöva omdefiniera några av de tekniker som tagits upp. Verktyget i sig är redan en form av redigeringsverktyg då vi skapar och ändrar på objekt i terminalsidor. Inom varje teknik kommer vi kort att ta upp hur de anpassar sig till visuell, textuell och modulär programmeringmetodik i Verktyget. Vi kommer använda orden objekt och modul för att beskriva noder.

För att få fram vilken teknik som ska användas väger vi dessa mot varandra genom frågorna (5.2.2). Tekniken implementeras aldrig och vi föreslår endast en specifikation för arkitekturen.

Resultat

Presenterar på nästa sida.

	SUS[5]	**Rank**
Modulär	76	6
Textuell	69	8

Tabell 1: Sammanfattade resultat från Tabell 2
SUS = Är medelvärdet av Tabell 2s SUS värden för respektive metodik.
Rank = Sammanslagna rank värdet.

NR	Modulär(SUS)	Textuell(SUS)	Rank	Bra	Dåligt
1	77,5 B	50 F	M/T	**Modulär:** Var lättare än textuellt. Man kunde se vad man skulle klicka i lättare än på textuella. **Textuell:** Bra med snabb hjälptext. Blev lättare desto mer man arbetade med det.	**Modulär:** - **Textuell:** Måste hålla rätt på detaljer.
2	80 B	82,5 A	T/M	**Modulär:** Lätt att se alla delar för en modul och vad man då behöver lägga till. Blev inte rörigt. Lätt använligt. **Textuell:** Inget fel med snabb hjälptext. Lätt förstårligt.	**Modulär:** Svårt att tänka sig vad IDna gör och vad de är för något. **Textuell:** Svårt att se vart man ska göra ändringar om man inte är van vid programmering.
3	62,5 E	77,5 B	T/M	**Modulär:** Vänjer man sig vid det går det fort att göra. Bra för folk som inte är inne i kod. **Textuell:** Man har full kontroll. Väldigt mycket enklare att få det överskådligt än det modulära när det blir mycket.	**Modulär:** Eftersom alla rutor ser lika ut blir det svår överskådligt när det blir väldigt många rutor. **Textuell:** Inte så bra för folk som inte vet hur man kodar. Finns ett lärande steg.
4	82,5 A	82,5 A	B	**Modulär:** Förvirrad först. Bra skit. **Textuell:** Förvånad över att knapparna var så bra. Bra skit. knapparna är bra för nybörjarna. **Annat:** Ändra mellan båda var väldigt bra! De kompliterar varandra väldigt bra.	**Modulär:** ID-taggarna var lite konstiga. Vore bra om man fick ändra dem själv. Text för ID vore bra. **Textuell:** Highlightingen. Helst svart bakgrund med ljusa färger. PosX, PosY olika färger.
5	77,5 B	52,5 E	T/M	**Modulär:** Det var enklare att använda. Lättare att förstå. Bra sätt att introducera programmering. **Textuell:** Kan tänka mig att det går snabbare. Fast vet inte.	**Modulär:** - **Textuell:** -

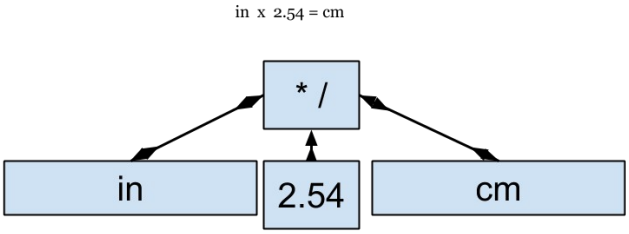
Tabell 2: Resultat ifrån HIFI undersökningen

Modulär(SUS) = SUS värde ifrån modulära delen **Textuell(SUS) =** SUS värde ifrån textuella delen

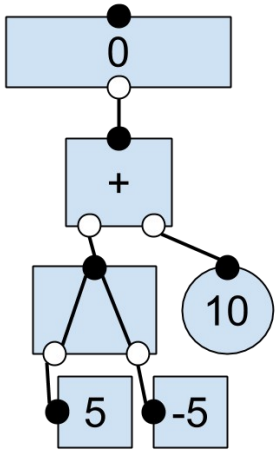
Rank = Rank frågan. Vilken vill ni använda helst. Ett språk som deltagare ville använda(ligger på vänster sida) fick två poäng. Det andra fick ett poäng och tyckte de om båda fick de ett poäng var. T = Textuell, M = Modulär och B = båda.

Bra = Vad tyckte deltagaren var bra med metodiken

Dåligt = Vad tyckte deltagaren var dåligt med metodiken



Figur 1: Fabrik exempel, enkel tum-cm konverterare



Figur 2: InterCONS exempel

BILAGA 3: EXTRA TEORI

Visuell programmeringsmetodik

Hils, Daniel D[18] tar upp 15 olika visuella programmeringsspråk bland dessa är Fabrik[19] och InterCONS[20] intressanta för sin inriktnig mot att skapa användargränssnitt. Ett Annat visuellt programmeringsspråk är Hammerwatchs Editorns[14] skriptspråk där utvecklaren skapar spelmiljöer och samtidigt scriptar händelser som spelarna upplever.

Fabrik

I Fabrik kallas noder för komponenter och kanter kallas för kablar. Komponenter är funktioner. En standardmängd av dessa erbjuds och programmerare kan även skapa egna. Standardmängden av komponenter hanterar aritmetik, texthantering, grafisk manipulering och innehåller komponenter som representerar gränssnittsobjekt samt en komponent för att rita egna bilder.

Utöver detta genererar även komponenter grafiska element som inkluderar rektanglar, ovaler, linjer, polygoner eller bitmappar. Med en annan grupp av komponenter kan vi ändra storlek, rotation och andra egenskaper hos grafiska element. Kommunikationen i Fabrik sker på så sätt att vi inte drar kablar direkt mellan olika komponenter, istället sker detta mellan pins på komponenterna. Pins har tre syften: de visar hur många kablar som kan kopplas till en komponent, vart en koppling kan göras på en komponent och om kopplingen är för indata, utdata eller båda (dubbelriktat). Indata representeras av en pil som pekar mot komponentens kärna, utdata av en pil från och dubbelriktat av en diamant. En kabel används för att representera dubbelriktat flöde

I Figur 1 kan vi se ett exempel av Fabrik-kod. Koden är förenklad på så sätt att vi inte tar med text-siffer konvertering mellan ”in”/”cm” och ”* /”. ”in” och ”cm” är inmatningsfält där användaren matar in tum respektive centimeter. Inmatade värden konventeras och skrivs ut i motsatt fält. ”2.54” är en konstant. Ovanför kan vi se formeln (in x 2.54 = cm) för konventering.

InterCONS

I InterCONS är noder funktioner som kallas för lådor och kanter kallas för länkar. Länkarnas uppgift är att bära data mellan lådor. Lådorna kan vara konstanter, variabler, text eller funktioner. Bland de fördefinierade funktionerna finns aritmetikoperationer, min, max och relations operationer. Fördefinierade lådor för användargränssnittselement som sliders, knappar och mätare finns också.

”Fan In” och ”Fan Out” är lådor som används för att klyva eller smälta ihop länkar. En ”Path Splitter” är en låda som fungerar likt en Om/Eller-sats. Om villkoret är sant så sänds inkommande data ut genom en av lådans två utgångar. Vid falskt sänds datan genom den andra utgången.

Språket tillåter även iterationer i from av cykler i programmets riktade graf. Varje cykel har minst en form av aktiverare inkopplad. I InterCONS är dessa knappar. När en användare trycker ner och släpper en knapp körs en cykel. Om knappen hålls ner körs cykeln upprepade gånger.

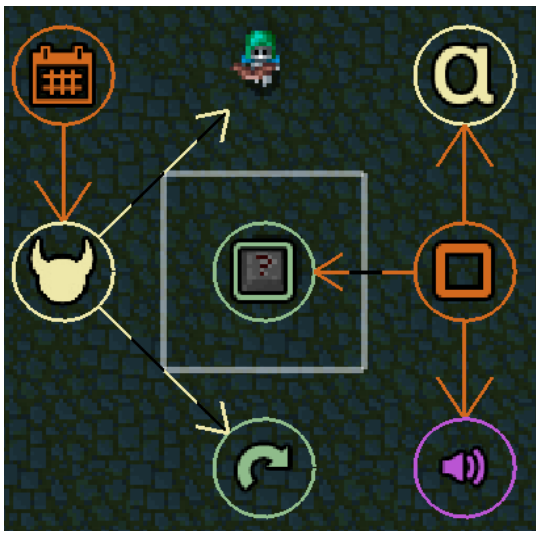
I Figur 2 kan vi se exempel på InterCONS kod. Vita cirklar är indata och svarta är utdata. Objekten med blå bakgrund är olika lådor. Cirkeln representerar konstanta värdet 10.

Högst upp har vi en låda som representerar en display. Indata och utdata är vad som visas för användaren.

Under har vi en plus-operator som tar in två länkar och slår ihop deras värden till ett och sänder resultatet vidare på utgångslänken. Datan måste vara numerisk.

Under plus-operatorn har vi en ”Fan In” som tar ingångslänkarnas flöde och låter det fortsätta på utgångslänken.

Längst ner har vi två knappar med värdet 5 och -5. När användaren trycker på någon av dessa kommer displayen visa 15 respektive 5. Hålls båda in kommer värdet blinka mellan båda.



Figur 3: Hammerwatch Editor Script exempel, ifrån spelets "Map Editor".

Hammerwatch Editor

Hammerwatch är ett ”fantasy hack-and-slash”-äventyrs-spel där användaren själv eller i grupp med upptill 3 andra deltagare spelar sig genom en karta full med skatter, monster och överraskningar. Vi kommer att koncentrera oss på spelets skriptspråk som finns i dess ”Map Editor” och används för att skapa olika händelser på kartor.

”Map Editorn” är annorlunda jämfört med Fabrik och InterCONS på så sätt att användaren direkt editerar den grafiska interaktionsmiljön och lägger sedan skriftspråket ovanpå. Vi delar upp ”Editorn” i två sidor:

- Kartan. Här hittar vi alla monster, hjältar, föremål, grafiska objekt samt kartans utseende.
- Skriptspråket. Här finns noderna som används för att bearbeta kartan samt kanter (pilar) som används för att koppla samman olika noder i riktade grafer.

Språket har ungefär 50 olika noder som används för att på olika sätt påverka kartan, sätta statiska variabler, spela upp ljud, visa texter eller andra aspekter. Vi har två olika former av kanter där den första är en enfärgad pil som visar aktiveringsflöde (datadriven) och den andra en färgad med svarta streck som visar efterfrågeflöde (efterfrågedriven) eller på annat sätt redigering av mål. I Figur 3 ses ett exempel som vi går igenom.

Den orangea kalendernoden i vänstra övre hörnet aktiveras när kartan laddas. Aktiveringen följer den orangea pilen nedåt till den vitfärgade figurnoden som vi ser har två streckade pilar. Den första pilen pekar på en figur och andra på en positionsnod. Vid aktivering av figurnoden beordrar den figuren att ta sig till positionsnoden. Detta leder till att figuren börjar förflytta sig och passerar då fyrkanten i mitten av bilden.

Den orangea noden till höger om mitten representerar en utlösarnod som lyssnar på den stora fyrkantiga noden i mitten av bilden. Detta kan vi se på grund av den streckade orangea pilen. När figuren som vi nämnt ovan rör sig in i området aktiveras utlösarnoden som då aktiverar övre och sedan undre noderna. Vi kan se kopplingen på bilden men inte aktiveringsordningen. Den skapas utifrån hur vi drar kanter under länkningsprocessen i ”Map Editorn”. I vårt fall drog vi först till den övre och sedan den undre.

Övre noden som visas som ett a i en cirkel kommer att skriva ut ”Hello World” i stor text på skärmen. Något som redigeras i ”Map Editorn”.

Undre noden som visas som en högtalare i en cirkel kommer att spela upp ett ljud. Val av ljudklipp sker i ”Map Editorn”.

Livscykel, faser och stadier

För att reda ut vilken av de olika programmerings-metodikerna som lämpar sig mest till grafisk gränssnitts-utveckling måste det finnas ett ramverk för att testa användbarhet och även se till så att utvecklingen sker mot användaren och inte utvecklaren. Det är väldigt lätt att anta att funktionalitet förstås av både utvecklare och användare, vilket inte alltid är fallet.

J. Gould[1] tror på att följa fyra principer i utveck-lingsprocessen för att uppnå en bra systemdesign:

- *Tidig fortsatt fokusering på användaren*, genom direktkontakt via interjuver, observationer, undersökningar eller gemensam design.
- *Tidig fortsatt användartestning*, där användaren utför riktigt arbete genom simuleringar eller prototyper för att samla in information om brister och styrkor i designen.
- *Iterativ design.* Beroende på information inför-skaffad från användartesterna ändras designen och detta steg upprepas.
- *Integrerad design.* Alla aspekter av användbarhet utvecklas parallellt och fokuserat på en punkt. Det vill säga att skisser av användargränssnitt, användarmanualer, språköversättningar och så vidare utvecklas samtidigt. En person ansvarar för utvecklingens riktning.

För att uppnå de fyra principerna bryter J. Gould ned arbetet i fyra faser.

- *Insamlingsfasen.* Här samlar vi information och skapar begreppsbildning.
- *Inledande designfasen.* Här bestämmer vi en preliminär specifikation för användar-gränssnittet.
- *Iterativa utvecklingsfasen.* Här undersöker vi mot testbara mål och fortsatt utvärdering av användbarheten.
- *System installationsfasen.* Här handlar det om att installera själva produkten hos användaren. Även om produkten löser många problem och är väldigt lättanvänd kommer personer inte att vilja

använda den om varan är komplicerad att installera.

Även om J. Gould har denna uppdelning av principer och faser är användbarhet fortfarande för abstrakt för att vi ska kunna studera det. J. Nielsen beskriver abstrakt fem användbarhetsmål i "The usability engineering life cycle. (1992)"[10]. F. Xavier sammanfattar dessa mål i "Usability basics for software developers."[4] från J. Nielsens bok ”Usability Engineering.(1993)”[13]:

- Lärbarhet, hur lätt det är att lära sig kärnsystemets funktioner.
- Effektivitet, hur många uppgifter per tidsenhet användaren kan utföra.
- Användaråterkoppling över tid, hur väl användaren kommer ihåg systemet efter en period av inaktivitet.
- Felfrekvens, hur många fel användaren gör när denne försöker utföra en uppgift.
- Tillfredsställelse, vad användarens intryck är av systemet.

En slutsats vi kan dra är att det är viktigt att ha en klar struktur av produktens utvecklingslivscykel. J. Nielsen[10] föreslår en modifierad och utbyggd version av J. Gould och C. Lewis[11] gyllene regler, vilka är; Tidigt fokus på användare och uppgifter, empirisk mätning och iterativ design som är föregångare till J. Goulds fyra principer.

Element i J. Nielsens[10] modell

Modellen är uppdelad i före design, design och efter design stadier. Dessa utförs stegvist i ordningen vi nämner. Viktigt att veta är att varje stadie har en eller flera metoder. En del av dessa kan utföras samtidigt och/eller upprepas. Vi behöver inte använda alla, däremot måste vi använda metoderna 7, 8 och 9.

Före designstadiet

1. *Lär känna användaren,* handlar om att samla information om användaren. Vem är användaren? Vilken uppgifter har denne? Var misslyckas användaren som produkten ska lyckas? Har användaren någon egen lösning?
2. *Konkurrentanalys.* Vilka nuvarande produkter löser redan existerande problem?
3. *Sätt upp användbarhetsmål.* Vilka av de attribut(se ovan) som tas upp ska vi koncentrera oss på? Vanligtvis är alla lika värda men som exempel är det större vikt på *lärbarhet* i ett system som ofta presenteras för nya användare.

Designstadiet

4. *Gemensam design.* Användaren är ingen professionell designer, däremot kan de reagera på föreslagna designer och ge intryck. De kan ställa frågor som designern inte tänkt på.
5. *Koordinerad design.* Designen måste vara konsistent mellan olika gränssnitt.
6. *Riktlinjer och heuristisk analys.* Designen måste följa riktlinjer som skapas av samhället och utvecklarna. Om produkten stöter på problem i utvecklingen rådfrågar utvecklare dessa.
7. *Prototyp.* Beskrivs i 4.2.2.
8. *Empirisk testning.* Utför undersökningar koncentrerade mot användaren, för att testa funktionalitet och designval.
9. *Iterativ design.* Med insamlad data från empirisk testning utvecklas produkten. Designval som användaren tycker om och fungerar behålls medan de som upplevs som ett hinder tas bort eller ändras. Detta kan involvera att designa om gränssnittet eller bara flytta på ett grafiskt objekt.

Efter design stadiet

10. *Samla användardata.* Vad gick bra respektive dåligt med designen? Vad kan vi ta med oss från projektet till nästa?

BILAGA 4: DIREKTAKTIVERING, SKRÄDDARSYLÅGE & TREDJEPART

Tekniker

Direktaktivering och skräddarsyläge

I Verktyget används direktaktivering för att ändra på positioner och beteenden för olika objekt. Därför slår vi ihop direktaktivering och skräddarsyläge. De kommer från och med nu kallas skräddarsyläge.

Visuell. Anvecklaren ser alla grafiska objekt som de existerar i Terminalen och kan via musen förflytta dessa och ändra på storleken. Övriga objekt presenteras som bubblor med en bild som representerar syftet. Mellan dessa dras pilar via musen för att bestämma händelseflöden. Genom att högerklicka på ett objekt med musen får anvecklaren tillgång till redigering. Redigering sker genom att anvecklaren ändrar på mindre kodblock.

Textuell. Anvecklaren ser alla objekt som kodblock. Jämfört med andra utvecklingsmetoder är vi alltid i skräddarsyläge och direktaktivering existerar endast i formen av redigering av attribut. Genom att antingen skriva över eller ändra koden kan anvecklaren ändra positioner och andra egenskaper hos ett objekt. Objekten kommunicerar mellan varandra genom att anropa olika IDn. De kan antingen aktivera eller hämta data.

Modulär. Anvecklaren ser alla objekt som modulära block. Hos dessa kan vi redigera egenskaper. Kommunikation mellan moduler sker med samma principer som hos textuell. Genom att aktivera en knapp på modulen kan vi ta oss till skräddarsyläge där vi kan redigera kodstycken för reaktion vid aktivering och returnering av data.

Tredjepart

Tredjepartsprogrammerare utvecklar nya objekt mot en standard som både Terminalen och Verktyget följer och kan liknas vid interface i Java. Ett objekt kompileras till en fil som sedan Terminalen och Verktyget kan läsa in. Programmeringsmetodiken som används här ska inte förknippas med de metodiker terminalsidor utvecklas med. På grund av avvikande tillvägagångssätt jämfört med de två tidigare kommer vi att gå in på de utmaningar programmerare möter när de utvecklar objekt för varje programmeringsmetodik. Viktigt att påpeka är att ett objekt inte kan vara direkt beroende av ett annat objekt. Varje modul sparas i var sin objektfil som lagras på hårddisken.

Visuell. Representationen av icke-grafiska objekt är en utmaning och kan skapa problem om flera personer arbetar med att ta fram samma objekt. Då skapas flera bilder och alla måste tolkas, vilket leder till ökad risk för misstolkning. Programmerare som grupp måste bestämma standarder för bilder så att inte missförstånd uppstår. Ett sådanfall skulle kunna vara att vi har två programmerare som utvecklar ett objekt för att hantera inloggning och använder olika bilder för att representera sitt objekt.

Textuell. Anvecklare måste bestämma exakt vilka taggar som finns och felhantering vid felaktiga eller avsaknade taggar. Dessutom måste de skapa dokumentation för taggarnas syntax.

Modulär. Programmerare måste bestämma exakt vilka egenskaper som ett objekt ska ha. De måste även välja någon form av kosmetisk taggning för att göra det lättare att skilja blockmodulen från andra block. Likt visuellt måste det tas fram någon form av standard mellan programmerana.

Frågor

Direktaktivering och skräddarsyläge

Vilken utvecklingsmiljö sker anvecklingen i?

Anvecklingen sker direkt i verktyget. I form av positionering- och värdeändringar.

Hur brant är inlärningskurvan?

Metoden kommer till anvecklaren naturligt. Redigeringen kan kopieras ifrån andra källor inom verktyget. Vilket leder till en låg inlärningskurva.

Hur lägger anvecklaren till nya moduler?

Anvecklaren kan inte lägga till moduler. Denne kan bara lägga in en existerande modul och skräddarsy den för att ändra vissa egenskaper, så som hur det anropar eller hanterar värden.

Hur lägger anvecklaren till ny funktionalitet?

Det går inte i denna form av anveckling att lägga in ny funktionalitet.

Hur ändrar anvecklaren på redan existerande moduler?

Anvecklaren kan inte ändra på standardmodulerna utan får använda en tidigare lösning. Se ovan under tillägg av nya moduler.

Hur tar anvecklaren bort moduler?

Anvecklaren kan välja att inte använda modulerna, men kan aldrig ta bort dessa.

Hur sprider anvecklaren skapade moduler?

Anvecklaren kan inte sprida den ”skapade” modulen, men kan däremot ge ut kod som skrivits i skräddarsyläge. Viktigt att notera är att även om anvecklaren skapar en knapp på en terminalsida måste denne skriva om koden på en annan. Detta leder till upprepade kodstycken.

Tredjepart

Vi kommer benämna anvecklaren som programmerare och anvecklare. Personer som utvecklar här är Tredjepartsprogrammerare.

Vilken utvecklingsmiljö sker anvecklingen i?

Programmerare väljer själv utvecklingsmiljö, men det måste vara för det språk vi använder för att utveckla systemet. Språket kommer benämnas som utvecklingsspråket nedan.

Hur brant är inlärningskurvan?

Programmerare måste ha kunskapen om utvecklingsspråket, APIet och hur objektfilerna lagras vid användning för både Terminal och Verktyg. Det tar även längre tid att utveckla objekt på grund av alla kringliggande kunskaper och utrustning, vilket leder till en hög inlärningskurva. Här lägger vi över mycket ansvar på yttre personer och lämnar oss sårbara för attacker.

Hur lägger anvecklaren till nya moduler?

Programmerare skapar en kodfil som följer utvecklingsstandarden och kompilerar filen till ett objekt som sedan lagras för Verktyget och Terminalen.

Hur lägger anvecklaren till ny funktionalitet?

Detta sker när programmerare lägger in ett nytt objekt. Viktig att notera är att anvecklare får enorm frihet för utveckling av egen funktionalitet. Däremot är säkerhetsaspekten inte inräknad, utan något som användare måste tänka på när de tar mot andras objekt.

Hur ändrar anvecklaren på redan existerande moduler?

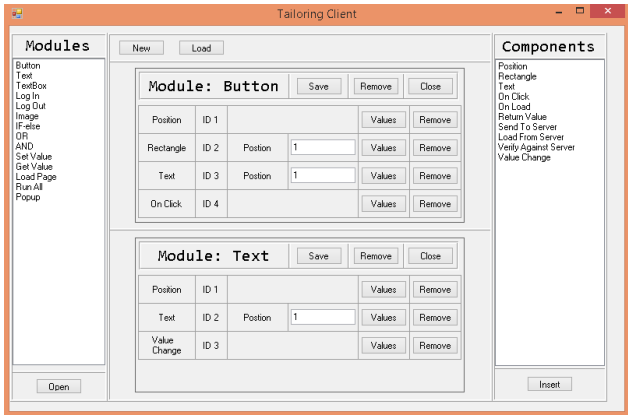
Programmerare kan antingen ändra på källkoden och kompilera en ny fil eller skriva över den redan existerande. Annars måste någon form av dekompilering ske.

Hur tar anvecklaren bort moduler?

Anvecklare raderar objektfilen ifrån hårddisken.

Hur sprider anvecklaren skapade moduler?

Programmerare sprider moduler som objektfiler. Dessa delas mellan utvecklare och lagras på hårddisken för Verktyget och Terminalen.



Figur 1: Skräddarsyklient

BILAGA 5: ARKITEKTURSPECIFIKATION

Anvecklingsverktyg

Skräddarsyklient är steget mellan skräddarsyläge och tredjepart. Det är varken för enkelt eller avancerat. Vi utövar redan en form av redigering i Verktyget. Därför ska KÄK använda Skräddarsyklient till Verktyget. När vi benämner skräddarsyklienten i bestämd form syftar vi på den applikation som KÄK ska använda.

Hur ska utvecklingsmiljön (Figur 1) se ut?

På vänster sida ska anvecklaren ha en lista med existerande moduler. På höger sida ska anvecklaren ha en lista med existerande komponenter. I mitten ska anvecklaren ha den/de nuvarande valda modulen/-rna. Över detta finns en menyrad med knappar och menyer. Varje modul kan sparas enskilt eller i grupp för att lättare kunna spridas mellan anvecklare. Dessutom ska en anvecklare kunna ladda in moduler med samma namn för att jämföra och redigera efter behov.

Hur lägger anvecklaren till nya moduler?

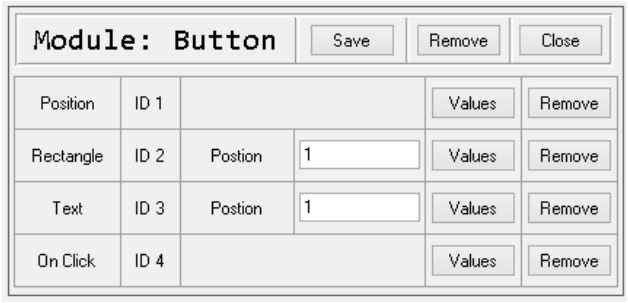
Genom att trycka på en knapp i menyraden får anvecklaren upp en guide som först frågar om namn sedan om den nya modulen ska kopiera en redan existerande eller vara tom.

Hur lägger anvecklaren till ny funktionalitet?

Genom skräddarsyläget för komponenter kan anvecklaren ändra viss funktionalitet hos en komponent, men bara om den ligger i en modul. Egenskaper som kan ändras är anropet mot server eller databas, lagring och hämtning av data och vad som händer vid aktivering av en modul. Skräddarsyläge nås genom att trycka ”Values” som du kan se i Figur 2.

Hur raderar/redigerar anvecklaren redan existerande moduler?

Anvecklaren väljer en modul på vänster sida som laddas in i mitten fönstret. Nu kan anvecklaren ta bort eller lägga till komponenter och ändra den inre kommunikationen. Se Figur 2 för exempel på modul. Anvecklaren kan radera en modul genom att trycka ”Remove” högst upp på modulen.



Figur 2: Exempel på knappmodul

Vilka komponenter ska finnas?

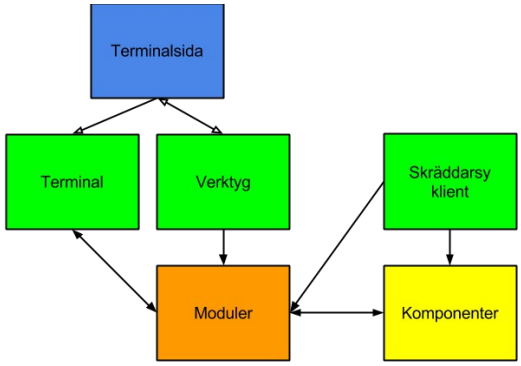
- Datatyper siffror, tecken, strängar, bilder och ljud.
- Events. T.ex. Så att användaren kan mata in text eller ”musklicka” på en komponent.
- Hämta, verifiera och sända till server.
- Byta terminalsida.
- Lagra och ta bort statiskdata för t.ex. vem är inloggad för tillfället.
- Komponenter som kan aktivera ett antal andra komponenter i följd från en lista.
- Logiska operationer på datatyperna. Icke, och, eller, antingen-eller, om-så samt om-och-endast-om. Jämförelse-operationer för siffror, tecken och strängar.
- Öppna och stänga mindre fönster i terminalen.
- Ändra och hämta värden i moduler under körning.

Arkitektur – Systemdelar

Ett system är en implementation av arkitekturen. Inre systemdelar är delar som är implementerade på samma kärna. Vad en kärna är tar vi upp i Kärnarkitektur.

Förklaring till Figur 3:

Blått är sparfil likt en HTML-fil.



Figur 3: Arkitekturöversikt

Grönt är en applikation.

Orange är sparade objekt som baserar sig på en klass. Klassen är en kärnklass.

Gult är klasser som ärver en basklass.

Pilarna representerar relationer mellan olika systemdelar.

Stängda pilar betyder inre systemberoenden, vilket betyder att hela systemet måste basera sig på samma kärna. D.v.s. att alla tre applikationer måste använda samma Modul- och Komponentklass. Pilarnas riktning anger beroende och kontrollflöde.

Öppna pilar betyder yttre systemberoenden, vilket betyder att systemdelen kan komma ifrån ett yttre system, men fortfarande tolkas av det inre systemet. Pilarnas riktning anger dataflöde.

Relationerna behandlar vi i 6.3.3.

Nedan förklarar vi varje systemdel.

Kärnarkitektur

Vid start av Terminal, Verktyg eller Skräddarsyklient laddar vi in alla moduler som återfinns i en katalog på hårddisken vilket bygger en lista där vi knyter modul namn till modulobjekt.

Om vi följer pillarna i Figur 3 till de systemdelar som är längst ner finner vi att Terminal, moduler och komponenter är längst ner. Detta leder till att dessa delar är kärndelar i systemet. Vilket betyder att de klasser som används här måste vara samma för ett system.

Terminalsida

En terminalsida är en textfil där vi finner den kod som representerar det grafiska gränssnittet. Filen är skirven i XML. Dess yttersta element heter ”root” och är en tagg som används för att knyta ihop allt. Inuti finner vi element som representerar olika moduler. Koden i filen följer samma standard som finns i undersökningen (textuell programmering). I de inre elementen sparar vi värden. Om vi t.ex. har en knapp sparar vi dess position, text och vilken modul den ska aktivera när användare ”musklickar” på objektet.

Terminal

Terminalen laddar in terminalsidor och moduler, men vet inget mer om vad som sker inuti modulerna. Anrop mellan moduler sker genom ID-anrop och kan då antingen aktivera eller hämta data. Hur moduler aktiveras från start beror på modulen, en del aktiveras vid inladdning av terminalsida och andra när användaren trycker på dess grafiska representation. Terminalens uppgift är att tolka terminalsidor och presentera deras grafiskagränssnitt.

Verktyg

Verktyget vet vilka moduler som existerar och vilka som finns på den nuvarande öppna terminalsidan. Här sköter vi kommunikationen mellan moduler och standardvärden som positionering, storlek och texter. Verktygets uppgift är att skapa terminalsidor och testköra dessa.

Skräddarsyklient

Skräddarsyklienten agerar inte på kommunikation mellan moduler, men vet att den existerar. Klienten fokuserar på de komponenter som utgör moduler och deras kommunikation. Det är dess uppgift att skapa, ändra och ta bort moduler samt fylla dessa med komponenter.

Moduler

Moduler är behållare för komponenter. Med hjälp av komponenter skapar vi t.ex. modulen knapp eller lista. Terminal, Verktyg och Skräddarsyklient arbetar mot olika sidor av moduler, vilket leder till att Moduler kommer ha ett större interface men bara vissa delar blir bearbetade av ett visst program. Moduler ligger i samma katalog på hårdisken för Terminal, Verktyg och Skräddarsyklient. Däremot kan Skräddarsyklient ladda in moduler från andra kataloger. För att se exempel på hur de olika applikationerna upplever en modul se Figur 4. Moduler sparas direkt som objekt på hårdisken. När vi skapar en modul i Verktyg eller Terminal kopierar vi en befintlig modul vi laddat in.

Komponenter

Det är den minsta delen i arkitekturen. Här finner vi behållare för data som positionering, grafiska former som cirklar och rektanglar och event för att på olika sätt kunna aktivera en modul som ”OnClick”. Komponenterna ligger sammlade i en fil och samma katalog som modulerna.

Arkitektur – Relationer

I Figur 3 kan vi se en arkitekturöversikt. Relationerna i Figur 3 beskriver vi nedan. Värt att notera att på grund av Terminalsida är en sparfil så blir relationerna mot systemdelen algortimer istället för APIer.

Verktyg ↔ Terminalsida

Verktyget kan spara och ladda terminalsidor.

Spara en terminalsida:

1. Utvecklaren bestämmer var Verktyget ska spara filen.
2. Verktyget skapar en temporär XML-fil på minnet och skriver in yttersta elementet ”root”.
3. Verktyget itererar igenom en lista med skapade moduler för terminalsidan. För varje modul skapar Verktyget ett element med samma namn som modulen och lägger in attributet ID med IDet ifrån modulen. Därefter sänder Verktyget elementet in i modulen så att den kan skriva ner viktig data.
4. Verktyget skriver den temporära XML-filen till en katalog på hårdisken bestämd av utvecklaren.

Ladda en terminalsida:

1. Utvecklaren bestämmer vilken fil som ska laddas. Filen laddas in på minnet i form av en XML-fil.

2. Verktyget itererar genom alla element i yttersta elementet ”root”. För varje element läser Verktyget in namnet och skapar en modul som är kopplad till namnet. Verktyget läser in attributet ID. Från detta sätter Verktyget in modulen i en lista efter ID. Verktyget skapar en panel A där högst upp skrivs modulens namn och id, samt en knapp för att ta bort modulen. Sedan skapar Verktyget ytterligare en panel B som sänds till modulen för att skriva ut vilka attribut utvecklaren kan ändra på. Panel B läggs på panel A och Verktyget sparar panelen. Till sist sänder Verktyget in elementet i modulen för att ladda in sista data.

Terminalsida → Terminal

Terminalen kan läsa in terminalsidor.

Ladda en terminalsida:

1. Filen som ska laddas in bestäms antingen av en modul under körning eller vid uppstart av Terminalen. Filen läses in i minnet som en temporär XML-fil. Från detta får Terminalen yttersta elementet ”root”.
2. Terminalen itererar genom yttersta elementet. För varje element läser Terminalen in namnet och skapar en modul som är kopplad till det namnet. Terminalen läser in attributet ID. Från detta sätter Terminalen in modulen i en lista efter sitt ID. Sen sänder Terminalen in elementet i modulen för att ladda in sparad data.
3. Terminalen itererar genom listan med moduler och för varje modul anropar applikationen på en initieringsmetod där Terminalen och en panel är parametrar. Om modulen har något att rita ut skrivs det på panelen.
4. Terminalen itererar genom listan med moduler och anropar inladdningseventet så att moduler som behöver aktiveras vid inladdning får det.

Terminal ↔ Moduler

Här beskriver vi interfacet mellan Terminal och modul. Vi börjar med att beskriva de metoder modulen behöver för att Terminalen ska kunna arbeta mot objektet:

- getName(): För att hämta namn på en modul.
- getID(): För att hämta IDet på en modul.
- copy(): För att kopiera en modul.
- activate(Event): För att aktivera en modul. Parametern ”Event” förklarar varför.
- getValue(Event): För att hämta data från en modul och parameter ”Event” förklarar varför.
- load(Element): För att ladda data från ett element. Parameter ”Element” innehåller elementet.
- initClient(Terminal, Panel): För att initiera en modul efter att alla moduler blivit laddade. ”Terminal”-parametern sänds med för att modulen ska kunna spara denne för senare bruk. ”Panel” sänds med för att modulen ska rita ut om behovet finns. Om modulen är en knapp sätter vi ut den likt vi har i Figur 4 till vänster på panelen.

Nedan beskriver vi de metoder Terminalen behöver för att moduler ska kunna arbeta mot den:

- activate(ID, Event): För att aktivera en modul vars ID är ”ID” och ”Event” är då varför vi aktiverar modulen.
- getValue(ID, Event): För att hämta data från en modul vars ID är ”ID” och parameter ”Event” berättar då varför.
- loadPage(FileName): För att ladda in en terminalsida med fil namnet som finns i parameter ”FileName”.
- setStatic(Name, Value): För att sätta en statisk variabel. Där parameter ”Name” innehåller namnet och parameter ”Value” innehåller värdet.
- getStatic(Name): För att hämta en statisk variabel. Där parameter ”Name” innehåller namnet på variabeln.
- clearStatic(Name): För att ta bort den statiska variabel. Där parameter ”Name” innehåller namnet på variabeln.
- hasStatic(Name): För att ta reda på om en variabel (vars namn finns i parametern ”Name”) existerar.

Figur 4: Terminal, Verktyg, Skräddarsyklient syn av en modul

- Statiska metoder finns för att kunna spara viktiga värden, t.ex. vilken användare är inloggad.

Verktyg → Moduler

Vi förklarar ovan i ”Verktyg → Terminalsida” hur en terminalsida laddas in, men vi går inte djupare in på de metoder som krävs. Relationen mellan Verktyg och modul är enkelriktad från Verktyg till modul. Däremot har modulen en specifik funktion för Verktyget.

- getName(): För att hämta namn på en modul.
- getID(): För att hämta IDet på en modul.
- copy(): För att kopiera en modul.
- save(Element): För att spara data till ett element. Parameter ”Element” innehåller elementet.
- load(Element): För att ladda data från ett element. Parameter ”Element” innehåller elementet.
- initEditor(Panel): För att initiera en modul efter att alla moduler blivit laddade. ”Panel” sänds med för att modulen ska rita ut de grafiska objekt som krävs ”för att kontrollera den”. Om modulen är en knapp sätter vi ut den likt vi har mitten i Figur 4.

Skräddarsyklient → Moduler

Relationen mellan Skräddarsyklienten och moduler är enkelriktad. Modulen behöver inte arbeta mot klienten. För att spara en modul i Skräddarsyklienten skriver vi ner objektet till en fil på hårddisken.

- getName(): För att hämta namn på en modul. Skräddarsyklienten ser namn som IDn för moduler.
- setName(Name): För att sätta namnet på en modul. Där parameter ”Name” är namnet.
- getComponent(ID): För att hämta en komponent från modulen som har samma ID som finns i parameter ”ID”.
- getComponents(): För att hämta en lista med alla komponenter som finns i en modul.
- addComponent(Component): För att lägga till en komponent som finns i parameter ”Component” och retunerar dess ID. Komponenten tilldelas ID av modulen.
- removeComponent(ID): För att ta bort en komponent med IDet som finns i parameter ”ID”.
- save(): För att spara ner allt innan vi skriver modulen till hårddisken.

Skräddarsyklient → Komponenter

Vid start av Skräddarsyklienten laddar vi in alla komponenter från en bibloteksfil. Komponenter behöver detta interface:

- getName(): För att hämta namnet på en komponent. Skräddarsyklienten ser namnet som ID.
- getID(): För att hämta ID på komponenten så att anvecklaren kan ta hand om inre kommunikation och klienten ska kunna skriva ut det (kan ses i Figur 2 och är då ”ID X”).
- copy(): För att kopiera komponenten.
- initTailoring(Panel): För att initiera en komponent på panelen som finns i parameter ”Panel”. Detta kan vi se i Figur 4 längst till vänster eller klarare i Figur 2. Texten position och textrutan är då inlagda av komponenten.
- initTailoringMode(Panel): För att initiera skräddarsyläge för en komponent. Detta sker när vi trycker på ”Values”, knappen kan vi se i Figur 2. Parameter ”Panel” är panelen i det nya fönster som öppnas.
- saveMode(): För att spara data vid stängning av Skräddarsyläge.

Moduler ↔ Komponenter

Relationen går åt båda hållen så vi kommer ta upp de metoder båda delar behöver för att arbeta mot varandra. Vi tar även upp transitiva användningar som att när en modul blir aktiverad i Terminalen aktiverar den en komponent vilket leder till att Terminalen aktiverar komponenten.

Komponens interface:

- activate(Event): För att aktivera komponenten. Parametern ”Event” förklarar varför. Används i Terminalen.
- getValue(Event): För att hämta data från en komponent och parameter ”Event” förklarar varför. Används i Terminalen.
- save(): För att spara ner allt innan vi skriver modulen till hårddisken, så att vi säkert vet att all data följer med. Används i Skräddarsyklienten.
- copy(): För att kopiera en komponent när vi kopierar en modul. Används i Terminal och Verktyget.
- save(Element): För att spara information till elementet i parametern ”Element”. Används vid sparning av en modul i Verktyget.
- load(Element): För att ladda information från elementet i parametern ”Element”. Används vid laddning av en modul i Verktyget och Terminalen.
- initClient(Terminal, Module, Panel): För att initiera en komponent i Terminalen. Används från modulens initClient(Terminal, Panel). ”Terminal”, används för kommunikation mellan

komponenten och andra moduler eller byta terminalsida.
”Module”, är ägaren av komponenten och används för inre kommunikation mellan komponenter.
”Panel”, om komponenten behöver rita ut något som en knapp.

- initEditor(Panel): För att initiera en komponent i Verktyget. Används från modulens initEditor(Panel). ”Panel” är panelen som komponenten skriver ut inmatnings rutor och liknande på.

Modulens interface:

- activate(ID, Event): För att aktivera en komponent i modulen vars ID är ”ID” och ”Event” är då varför vi aktiverar modulen.
- getValue(ID, Event): För att hämta data från en komponent i modulen vars ID är ”ID” och parameter ”Event” berättar då varför.