Användbarhetsaspekter för domänexperter inom området programmering vid användning av grafiska verktyg

(HS-IDA-EA-02-505)

Anna Kardell (a98annka@student.his.se)

Institutionen för datavetenskap
Högskolan i Skövde, Box 408
S-54128 Skövde, SWEDEN

Examensarbete på det kognitionsvetenskapliga programmet under vårterminen 2002.

Handledare: Anna-Sofia Alklind Taylor

Användbarhetsaspekter för domänexperter inom området programmering vid användning av grafiska verktyg

Examensrapport inlämnad av Anna Kardell till Högskolan i Skövde, för Kandidatexamen (B.Sc.) vid Institutionen för Datavetenskap.

2002-06-07

Härmed intygas att allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och att inget material är inkluderat som tidigare använts för erhållande av annan examen.

Signerat: _____

Användbarhetsaspekter för domänexperter inom området programmering vid användning av grafiska verktyg

Anna Kardell (a98annka@student.his.se)

Sammanfattning

Denna studie har avsett undersöka hur ett lämpligt gränssnitt för en expert ska se ut. En expert utvecklar enligt Soloway, Adelson & Ehrlich (1988) kognitiva scheman vilka används vid problemlösning. När dessa scheman inte stämmer, och inte kan användas, sjunker expertens prestation till samma nivå som hos en novis. Detta gör att när man utvecklar gränssnitt för experter måste man ta hänsyn till den kunskap som experten besitter. I studien genomfördes en observation där man lät sex programmerare utföra uppgifter i tre befintliga program. Av detta kunde man dra slutsatserna att experter vill ha överblick över koden när de programmerar, samt att det är viktigt att använda etablerade begrepp i gränssnittsdesignen.

Nyckelord: Användbarhet, Experter, Programmering

Innehållsförteckning

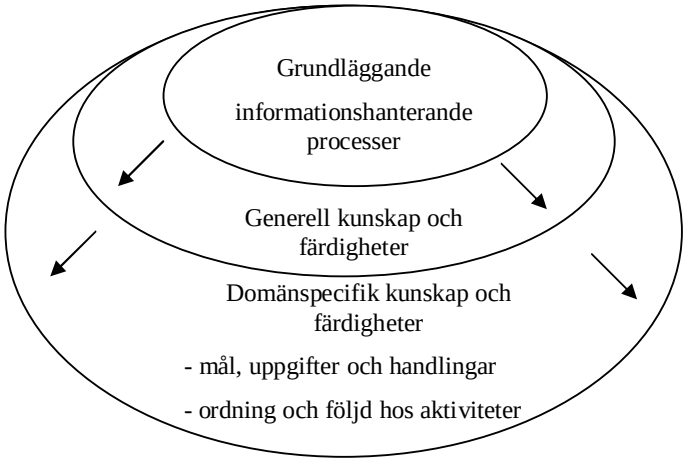
Referenser

Bilagor

1 Inledning

När man skapar ett system är det väldigt viktigt att man tar hänsyn till de förväntade användarna. Vilka kunskaper har de? Vilka är deras mål när de använder systemet? Olika användare har olika egenskaper och därför olika behov i ett system. Detta arbete kommer att fokusera på användarens kunskap. Hur påverkar användarens tidigare kunskaper användningen av ett nytt system?

När en användare använder ett system behöver den kunskap. Ju mer kunskap en användare har desto mer ökar dess kapacitet. Detta illustreras av Figur 1 nedan:



Figur 1. De egenskaper hos användaren som är viktiga i användningen av program (efter Kujala & Mäntylä, 2000, s.2).

Figur 1 beskriver de egenskaper som är viktiga för användaren i användningen av ett system. Den inre nivån visar de grundläggande informationshanterande processer som människan har. Den mellersta nivån visar generell kunskap och färdigheter vilka individen kan besitta. Den yttersta nivån visar domänspecifik kunskap och färdigheter. Pilarna visar hur individens kapacitet ökar när kunskapsnivån ökar. Ju mer specifik kunskap som användaren har desto större blir dess kapacitet.

Detta arbete ska handla om den kunskap som finns i den yttre nivån, det vill säga domänspecifik kunskap. Enligt figuren så ökar individens förmåga ju mer kunskap denna har. Men stämmer detta alltid? Är en expert alltid bättre än en novis inom den aktuella domänen? För att nå en nivå av expertis krävs oftast flera års träning (Glaser & Chi, 1988). Under denna träning utvecklar experten sin förmåga att strukturera informationen inom området. När man utvecklar ett system för dessa experter måste man därför ta hänsyn till expertens redan befintliga kunskapsstrukturer. Om man inte gör detta, kommer experten inte kunna dra nytta av sin kunskap och hans prestation blir den samma som hos en novis. Exempel på detta finns

inom flera andra domäner, däribland schack (Chase & Simon, 1973) samt programmering (McKeithen, Reitman, Reuter & Hirtle, 1981)

Detta arbete kommer att rikta in sig på en specifik grupp experter, nämligen expertprogrammerare. Tidigare studier av expertprogrammerare av Soloway, Adelson & Ehrlich (1988) har visat att experter har svårare att förstå ett program som inte följer expertens invanda konventioner än de program som följer dessa. Noviser däremot, har lägre kunskap inom programmering och har därför inga invanda konventioner. Deras prestation skiljer sig därför inte lika mycket mellan de båda betingelserna. Enligt detta kan man dra slutsatsen att det är viktigt att expertprogrammerarna kan följa sina invanda konventioner, för att kunna prestera bra.

Vad innebär då detta för användbarhet? Enligt Kujala och Mäntylä (2000) beror hur väl användaren kan använda ett system på hur väl detta system matchar användarens mål och handlingar, och hur väl användaren kan använda sin kunskap om uppgifter och procedurer. Följaktligen kan inte användaren använda ett system väl, om detta system inte stödjer användarens kunskapsstrukturer.

Hur ska ett system vara utformat för att ta tillvara på experters kunskap? Enligt Dillon och Watson (1996) är detta ett område som bör utforskas mer. Det har forskats mycket om att dra generella slutsatser om gränssnittdesign utifrån förväntade användare, men resultatet har inte kunnat visa några andra resultat än väldigt generella designriktlinjer, som exempelvis: experter föredrar kommandospråk och noviser föredrar menyer (Dillon & Watson, 1996). Detta arbete kommer att undersöka hur väl de tre programmen VAPS, GL Studio samt 3ds max stödjer expertprogrammerarnas tidigare kunskap inom programmering, och på så sätt underlättar deras arbete i dataprogrammet.

Eftersom de tre programmen besitter väldigt olika egenskaper, förväntas resultatet visa på egenskaper som är viktiga i gränssnittsdesign för expertprogrammerare. Då arbetet är utforskande förväntas dock dess viktigaste bidrag inom området bli en rad frågor vilka kan leda fram till ny forskning inom området.

2 Bakgrund

Det är viktigt att ta hänsyn till användarnas kunskaper. Hur konstruerar man då ett system för en användare som redan besitter en mängd kunskap, exempelvis experter?

2.2 Expertkunskap

Experter är en användargrupp som har undersökts flitigt genom åren. Skillnaden mellan experter och noviser har undersökts i olika sammanhang, bland annat: schack (Chase & Simon, 1973), programmering (McKeithen, et al., 1981; Soloway, et al., 1988), fysikproblem (Chi, Feltovich & Glaser, 1981) samt maskinskrivning (Gentner, 1988).

2.2.1 Definition av expertis

Inom Människa-Dator Interaktion (MDI) är det vanligt att likställa datorvana med expertis. Prümper, Zapf, Brodbeck & Frese (1992), exempelvis, lät expertis bestämmas av hur länge en försöksperson arbetat med en dator, hur många program denna kunde, samt hur många timmar om dagen försökspersonen arbetade med datorer. Datorvana är dock inte alltid liktydigt med expertis. Datorn är idag ett verktyg som används i en rad olika sysslor och domäner. Många personer använder idag datorer i sitt dagliga arbete, och använder många olika program varje dag, utan att för den skull vara experter på datorer. Relevant vore istället att titta på en specifik syssla utförd på datorer, till exempel programmering. Vad som definierar en expert är till stor del beroende på vilken domän man talar om. Hoffman, Shadbolt, Burton och Klein (1995) anser dock att det är viktigt att kunna operationalisera expertis, det vill säga finna mätbara kriterier för vad som är en expert. De ger följande förslag till en definition:

> The distinguished or brilliant journeyman, highly regarded by peers, whose judgements are uncommonly accurate and reliable, whose performance shows consummate skill and economy of effort, and who can deal effectively with rare or tough cases. Also, an expert is one who has special skills or knowledge derived from extensive experience with subdomains (Hoffman et al., 1995, s. 132).

Denna definition är lite för smal för att passa denna studie. Med en sådan definition kan det vara svårt att hitta lämpliga försökspersoner. Prümper et al.s definition däremot är för bred, eftersom många personer använder datorer i sitt dagliga arbete utan att vara experter. Denna definition bör istället specificeras kring vilken specifik syssla som utförs med datorn. I detta arbete kommer expertis definieras med hjälp av hur stor vana experterna har av programmerning. En programmeringsexpert bör ha fler års vana av programmering, samt arbeta med programmering regelbundet. Denna definition är väldigt bred, men gör det lättare att hitta försökspersoner.

2.2.2 Utveckling av expertis

Som visats ovan kan det vara svårt att definiera expertis. En sak som är gemensam för samtliga definitioner är att det tar väldigt lång tid att utveckla expertis. Hoffman, et al. (1995) föreslår ca 10 år som riktmärke för att utveckla expertis inom mer komplexa områden, exempelvis medicinska diagnoser. För att förstå vad det innebär att vara expert så är det intressant att veta vad som händer under denna långa utveckling av expertis. Att förstå

utvecklingen av expertis är viktigt för att få en bra inblick i området expertis, och kan hjälpa till att svara på frågan om vad som förändras hos människans kognitiva system när denna utvecklar expertis. Därför kommer nu en redogörelse av olika teorier inom detta område.

Fitts teori

Fitts (1964, i Proctor & Dutta, 1995) beskriver utvecklingen från novis till expert i tre faser. Dessa utgår ifrån att vi använder olika kognitiva processer i olika faser av vår inlärning. Den första fasen kallar Fitts den kognitiva fasen. I denna fas måste den som lär sig använda sina kognitiva processer för att förstå uppgiften och hur den ska utföras. De måste ge akt på utvändiga ledtrådar, de instruktioner som ges samt eventuell feedback angående prestation. I denna fas finns stor inblandning av medvetna kognitiva processer, därav namnet.

När instruktionerna och det förväntade målet har förståtts kommer personen in i den associativa fasen. I denna fas kopplas input samman med lämpliga handlingar, och behovet av verbala instruktioner minskar. I denna fas minskar antalet fel, men det tar även längre tid att lösa uppgiften.

Den tredje fasen, den autonoma fasen, markeras av minskad störning från externa krav samt minskat behov av uppmärksamhet. När prestationen har nått till den autonoma fasen sägs den vara automatisk och kräver inte längre medveten kontroll. Enligt Fitts kan det ta flera månader eller år att nå denna automatisering av uppgifterna. Denna automatisering gör att den som utför sysslan kan fokusera sin uppmärksamhet någon annanstans och därmed utföra flera olika sysslor samtidigt.

Utvecklingen genom dessa tre faser sker gradvis, och inte genom abrupta skiften mellan dessa.

Andersons teori

Anderson (1982, i Proctor & Dutta, 1995) har baserat sin modell på Fitts tre faser, men har lagt större vikt vid vilka kognitiva processer som ligger bakom varje nivå, och överföringarna mellan dessa. Han beskriver oftast utvecklingen av kunskap i två faser: deklarativa och procedurella, som motsvarar Fitts kognitiva respektive autonoma faser. I stället för att ha en separat fas som motsvarar Fitts associativa fas, så beskriver han en process som han kallar ”kunskaps insamling” (knowledge compilation) där kunskap konverteras från deklarativ form till procedurell form.

Skillnaden mellan deklarativ och procedurell kunskap beskrivs som fundamental i Andersons modell (Proctor & Dutta, 1995). Deklarativ kunskap är den kunskapsbas med fakta som en person har, medan procedurell kunskap är de uppgifter (skills) som en person kan utföra.

I den deklarativa fasen samlas instruktioner och information om situationen. Dessa fakta måste sedan repeteras och behållas i arbetsminnet för att kunna användas av våra kognitiva tolkningsmekanismer. När personen övar kommer vissa procedurer som är specifika för

uppgiften att utvecklas. Dessa kommer inte kräva tillgång till den deklarativa kunskapen om hur man utför uppgiften. Dessa procedurer beskrivs som ”om –så” regler. Exempelvis: *Om* detta villkor är uppfyllt *så* utför följande uppgift.

Utvecklingen till den procedurella fasen sker gradvis i och med användningen av ovan nämnda procedurer. Detta sker genom en process som Andersen kallar för ”finjustering” (tuning), som innebär att de regelbaserade procedurerna förfinas genom *generalisering*, det vill säga att tillämpa reglerna på flera olika situationer, *diskriminering*, att begränsa reglerna så att de endast gäller för passande situationer, samt att *förstärkning*, stärka de väl fungerande reglerna och göra de dåligt fungerande reglerna svagare.

Andersons schema beskriver liksom Fitts en gradvis förändring. Det kan i Andersons modell vara svårt att avgöra vilka som är experter och ej, eftersom en person som har tränat tillräckligt för att komma till den procedurella fasen förmodligen inte presterar lika bra som en expert. Enligt Proctor och Dutta behöver modellen förfinas mer för att det ska gå att avgöra vad som är karaktäristiskt för en expert.

Rasmussens teori

Rasmussen (1983, i Proctor & Dutta, 1995) har ett tredje ramverk för hur kognitiva färdigheter utvecklas. Hans ramverk riktar främst in sig på olika sorters beteende i olika steg av utvecklingen. De beteenden som uppvisas i de olika stegen av utvecklingen är kunskapsbaserat, regelbaserat och färdighetsbaserat beteende. Dessa kan liknas vid Fitts tre faser. Kunskapsbaserat beteende motsvaras av den kognitiva fasen, då utförandet av sysslan görs med mental kontroll samt verbal hjälp. Skillnaden mellan dessa är dock att i Fitts modell så är den kognitiva fasen en tidig del av utvecklingen av färdigheten, men i Rasmussens teori är det kunskapsbaserade beteendet hur individen presterar i detta steg av utvecklingen (Proctor & Dutta, 1995). Det regelbaserade beteendet styrs också av medveten kontroll, men sker efter regler, som i Andersons modell. Färdighetsbaserat beteende är det sista steget. Här är beteendet automatiserat.

Dessa tre modeller visar hur en färdighet utvecklas, och stämmer överens med forskning som beskriver skickliga maskinskrivare. Dessa utvecklar hög grad av automatik i sin syssla och kan ägna tankarna åt annat, till exempel dagdrömmar eller stavningen i dokumentet de renskriver (Gentner, 1988).

2.2.3 Skillnader mellan experter och noviser

Eftersom expertis är en utvecklingsprocess kan vi inte få den kunskap vi behöver bara genom att studera experter. Vi behöver jämföra dessa med personer som befinner sig i andra steg i sin utveckling, mellananvändare och noviser. Inom expertisforskningen studeras därför ofta skillnaderna mellan experter och noviser. Hoffman et al. (1995) kallar jämförelsen expert – novis ett paradigm inom expertisforskningen. Denna forskning har visat att det finns vissa generella egenskaper som skiljer experterna från noviser. Man har funnit gemensamma drag hos experter inom flera olika domäner (Glaser & Chi, 1988). Glaser och Chi räknar upp följande egenskaper:

- Experter är endast experter i sina egna begränsade miljöer.
- Experter kan uppfatta komplexa meningsfulla mönster i sin domän.
- Experter är snabbare än noviser på att utföra uppgifter inom sin domän och på att lösa problem.
- Experter har bättre kort- och långtidsminne inom sin domän.
- Experter ser och representerar problem på en djupare och mer fundamental nivå än noviser, vilka tenderar att representera problem på en mer ytlig nivå.
- Experter spenderar mycket tid till att analysera problem på en kvalitativ nivå.

Vi ska titta närmare på de för problemområdet viktigaste av dessa skillnader. Vad beror de på?

Experters förmåga att uppfatta komplexa mönster i sin domän:

Experter verkar kunna se med en blick vilken information som är viktig. Schackspelare uttrycker det som att de ”ser” vad som är rätt drag i en viss situation (Chase & Simon, 1973). Detta innebär dock inte att dessa har någon slags överlägsen perceptuell förmåga. Istället visar detta på experternas organisation av sin kunskap (Glaser & Chi, 1988). Experter kan ”chunka” informationen i stora meningsfulla enheter, vilket noviser inte kan.

Ett exempel på detta är Chase och Simons (1973) studie av schackspelare. Chase och Simon lät försökspersonerna göra två olika uppgifter. Den första var en perceptionsuppgift där försökspersonerna fick rekonstruera positionerna på ett schackbräde till ett eget schackbräde medan de fortfarande såg det första. Man undersökte här på hur länge försökspersonerna tittade på originalbrädet. Chase och Simon förutsatte att försökspersonerna skulle koda en ”chunk” per blick på brädet, och sedan rekonstruera denna på det egna schackbrädet. Man mätte sedan hur lång tid det tog mellan när en schackpjäs och nästföljande pjäs placerats på brädet. Korta intervall mellan pjäserna skulle visa att dessa ingick i samma ”chunk” i försökspersonens minne. De långa eller korta pausarna mellan pjäserna analyserades för att få information om hur pjäserna ”chunkades” perceptuellt. Resultatet visade att de erfarna spelarna hade kortare intervall i de situationer där de tittade på originalbrädet mellan utplaceringen av pjäserna. Det fanns inga skillnader mellan experterna och noviserna gällande tidsintervallen mellan de pjäser som ställdes ut på brädet utan att försökspersonen såg på originalbrädet emellan.

Uppgift nummer två var en minnesuppgift där man lät försökspersonerna se en schackuppställning i fem sekunder. Sedan skulle de rekonstruera positionerna på det egna brädet. De fick ställa upp så många pjäser de kom ihåg, och sedan fick de titta på orginal-

brädet igen. Chase och Simon räknade hur många gånger försökspersonerna behövde titta på brädet. Schackmästarna var betydligt bättre än noviserna.

Chase och Simon kom fram till att schackmästarnas överlägsna prestation att kunna rekonstruera meningsfulla schackpositioner finns i deras förmåga att se struktur i sådana positioner, och koda dem till ”chunks”. Noviser däremot tros koda informationen i flera mindre delar eftersom de inte har kunskapen som behövs för att lagra större enheter. Detta gör att noviser överbelastar arbetsminnet. I enlighet med detta faller minnesprestationen hos mästarna till samma nivå som hos noviser när positionerna som ska återgivas består av slumpmässigt placerade schackpjäser.

McKeithen et al. (1981) menar att experter inte bara organiserar sin kunskap i ”chunks”, utan dessa chunks är dessutom organiserade på visst sätt beroende på expertis. Experter strukturerar enligt McKeithen et al. kunskapen i hierarkiska strukturer. För att testa detta fick försökspersoner i form av programmeringsexperter och noviser göra ett minnestest liknande det Chase och Simon utfört. Försökspersonerna fick se programkod på ett papper, som antingen var logisk, eller med programraderna slumpmässigt blandade. Här fick man samma effekt som Chase och Simon, det vill säga experterna var betydligt bättre än noviserna på att återge de logiska programmen, men sjönk till novisernas nivå vid återgivning av de slumpmässigt sammansatta programmen.

För att testa om experterna hade en bättre organisation av sitt minne lät man sedan försökspersonerna memorera programmeringskommandon. När dessa återgavs av försökspersonerna lade man märke till vilka som oftast förekom tillsammans. Med hjälp av dessa data kunde man sedan skapa en modell för hur personerna organiserade dessa kommandon. McKeithen et al. fann att noviser organiserade orden efter första bokstav eller längd, och ibland dess likheter i betydelse inom naturligt språk. Hos experterna däremot dominerade erfarenheterna av programmering, och begreppen grupperades enligt dess funktion i programmeringsspråket. McKeithen et al. (1981) visar alltså att det finns en korrelation mellan graden av expertis och mental organisation av koncepten inom expertområdet.

Experter är snabbare än noviser på att utföra uppgifter inom sin domän och på att lösa problem

När det gäller sysslor som exempelvis maskinskrivning så beror experternas snabbhet på att de genom långvarig träning uppnått en hög grad av automatisering av uppgiften. Detta frigör minne så att de kan behandla andra aspekter av sysslan i stället (Gentner, 1988). De är snabbare därför att de har mer kapacitet för att genomföra sysslan. När det gäller problemlösning finns det en annan förklaring till experternas snabbhet. Experterna har genom sina interna scheman redan byggt upp lämplig respons på ett problem (Glaser & Chi, 1988). Dessa kan jämföras med Andersons ”om –så” regler (se kapitel 2.2.2). Om problemet ser ut på ett speciellt sätt utlöser det en viss respons. Här kan man också observera vikten av att experters kognitiva scheman stämmer vid problemlösning då de annars ger fel respons på ett problem.

Experter ser och representerar problem på en djupare och mer fundamental nivå än noviser

I Chi et al. (1981) studie i kategorisering av fysikproblem lät man experter och noviser sortera ett antal fysikproblem. Man fann att experterna sorterade problemen efter vilka grundläggande fysiska lagar problemet kunde hänvisas till, medan noviserna kategoriserade problemen efter de föremål som funnits med i problemformuleringen. Detta förklarar man med att experterna i sin analys av problemen aktiverar för problemen aktuella scheman.

Här kan schemateori vara värt att nämna. Kalyuga, Chandler och Sweller (1998) beskriver schema som en kognitiv struktur som möjliggör för människor att behandla flera delar information som en enskild del. Information strukturerad i scheman underlättar för korttidsminnet då det kan behandla många olika element som ett, och därav minska den kognitiva belastningen.

Enligt Soloway et al. (1988) använder expertprogrammerare scheman när de planerar sin programmering. De menar att experter har två sorters kunskap som noviser inte har. Dessa är ”programmeringsplaner” samt ”regler om programmeringskonventioner”. Med programmeringsplaner menar Soloway et al. programfragment som representerar vanliga händelsesekvenser i programmering. Exempel på en sådan händelse kan vara hur man utför en iteration. Programmerarna har då en ”Utför en iteration” plan, eller en ”Gör en loop” plan som innehåller information om hur denna handling utförs. Med regler om programmeringskonventioner menar Soloway et al. regler som specificerar konventioner som finns inom programmering. Exempel på detta kan vara att namnet på en variabel ska stämma överens med dess funktion. Dessa regler skapar förväntningar hos programmeraren om vad som bör finnas i programkoden. Dessa regler samt programmeringsplaner menar Soloway et al. (1988) stämmer överens med teorier om kognitiva scheman. De presenterar följande definition: Scheman är en allmän struktur som guidar personens tolkningar, slutledningar, förväntningar och uppmärksamhet (Graesser, 1981, i Soloway, et al., 1988). De menar att när ett program inte stämmer överens med de regler och planer som finns i expertens scheman så bryter programmet mot expertens förväntningar. Experterna får därför svårare att förstå dessa program. Noviser däremot, har lägre kunskap inom programmering och har därför inga invanda konventioner. De är därför inte lika känsliga som experter för brott mot programmeringskonventioner.

2.3 Experter och användbarhet

Enligt den tidigare forskningen (se kapitel 2.2), verkar nyckeln till experternas prestationer ligga i deras minne. De har inte bara mer kunskap, utan den kunskap de har är också bättre strukturerad (McKeithen et al., 1981). Men hur ska ett gränssnitt vara utformat för att man ska kunna dra nytta av expertens kunskap? (Med begreppet gränssnitt menas i detta arbete det grafiska användargränssnittet (GUI), det vill säga skärmbilden.)

Inom MDI har man studerat skillnader mellan experter och noviser länge. Till exempel har man visat att kunniga och vana användare föredrar kommandospråk framför menynavigering därför att kommandospråket i många fall är snabbare. Noviser eller ovana användare kan finna det svårt att behålla syntaxen i minnet (Streitz, Spijkers & Van Duren, 1987). Främst har

forskningen inom detta område behandlat individuella skillnader som kan finnas hos användarna och vilka implikationer de har på gränssnittsdesign. Enligt Mayhew (1992) finns det två olika sorters kunskap som är viktig att ta hänsyn till i designen av gränssnitt. Dessa är ”task experience” och ”system experience”. Med ”task experience” menas kunskap om uppgiften som är tänkt att utföras i systemet. Med ”system experience” menas kunskap om det givna systemet. De användare som är aktuella i undersökningen, som presenteras längre fram, har ingen vana av systemet, och inte heller av uppgiften som ska utföras. De har däremot en stor ämneskunskap inom domänen. Mayhews riktlinjer för gränssnittsdesign verkar därför inte vara applicerbara på detta fall. Men hur ska vi då veta huruvida ett gränssnitt är anpassat efter den kunskap som användaren har?

2.3.1 Användbarhet

En av de viktigaste aspekterna inom gränssnittsdesign är användbarhet. Syftet med att anpassa gränssnitt efter experter är att gränssnitten ska vara användbara. Vad är då användbarhet?

Användbarhet har definierats på många olika sätt. Här redogörs för ISO-definitionen. ISO-definitionen är gjord med tanke på att det inte går att definiera några attribut som kan garantera användbarhet i en produkt. Här har man därför inte ställt upp några generella kriterier som ett system måste uppfylla, exempelvis: ”systemet ska ha menyer”, istället har man baserat definitionen på hur bra produkten är när den används av en viss specifik användargrupp: ”**usability**: the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use” (ISO 9241-11, i Bevan, 1997, s.4).

Med verkningsfullhet (effectiveness) menas hur väl användarna lyckats genomföra avsedda uppgift. Här mäts alltså om systemet kan göra det man vill. Effektivitet (efficiency) är ett mått på hur mycket resurser som måste spenderas för att nå målet. Ett exempel på en sådan resurs är tid. Här mäts alltså hur väl systemet gör det man vill. Satisfaction är ett subjektivt mått på hur väl användaren accepterar produkten (Bevan, 1997).

Enligt denna definition kan utläsas att ett system för att vara användbart inte bara behöver utföra den tänkta uppgiften utan även göra det på ett bra sätt. Här har även lagts vikt vid att det är en viss specifik användargrupp som definitionen avser. Enligt denna definition så är det alltså nödvändigt att ett system är anpassat efter de tänkta användarna, experterna, för att vara så effektivt som möjligt. Och är systemet inte effektivt så har det följaktligen lägre användbarhet.

2.3.2 Domänexpertis

Många av de designriktlinjer som finns avseende experter och noviser syftar på experter på datorer. Men intressantare är att veta hur ett gränssnitt för domänexperter ska se ut. Gulliksen och Sandblad (1996) menar att nyckeln till att designa applikationer för en specifik domän är att förstå hur aktiviteterna i domänen utförs. Men vad är då en domän? Gulliksen och Sandblad definierar en domän som ett set av aktiviteter som ger samma effekt på användarna och deras prestation oberoende av interaktion med eventuella kunder, beslutsfattande med

mera. Ett exempel på detta är personaladministration. Där är sysslorna i stort sett de samma oberoende av vilken anställd som uppgiften berör.

Gulliksen och Sandblad (1996) menar också att användargränssnittet måste vara transparent. Detta innebär att användaren kan se hur systemet fungerar, istället för att ödsla kraft på att hantera gränssnittet. I övrigt kan Gulliksen och Sandblad inte ge några generella riktlinjer för hur domänspecifika gränssnitt bör utformas, då detta är beroende av domänen. De menar istället att man bör skapa en domänspecifik styleguide.

2.4 Dynamiska funktionsscheman

Detta arbete behandlar området domänexpertis. För att kunna studera området behövs en domän. Den domän som kommer att undersökas är dynamiska funktionsscheman.

I dagens flygplan är de tekniska systemen mycket komplicerade. De är uppbyggda i olika delsystem och spridda i flygplanet, utan fysisk koppling. Detta gör att dessa system blir mycket svåra att studera rent fysiskt. Det är, bland annat i utbildningen av flygtekniker, viktigt att kunna ge en korrekt överblick över systemen samt hur de samverkar. Därför skapar man dynamiska funktionsscheman. Ett dynamiskt funktionsschema är en grafisk applikation som består av ett principschema över flygplanssystemet, samt av dynamiska objekt såsom ventiler, strömställare regulatorer med mera. Dessa scheman är tänkta att kunna kopplas till en simulator. När en förändring sker i simulatorn ska detta kunna speglas på funktionsschemat. (Bo Andersson, personlig kontakt, mars, 2002)

Dynamiska funktionsscheman utvecklas idag med ett verktyg som heter VAPS. VAPS tillverkas av företaget Virtual Prototypes. I detta arbete har version 5.2 använts. Med VAPS kan man skapa funktionsscheman samt bygga upp kabinmiljöer och liknande. Men nu har många andra verktyg dykt upp på marknaden, vilka inte är speciellt konstruerade för att skapa funktionsscheman, men som kan lösa uppgiften på ett bra sätt. Företaget AerotechTelub vill därför utvärdera VAPS tillsammans med två andra alternativa program för att avgöra vilket som är bäst lämpat för uppgiften. Dessa två program är GL Studio och 3ds max. GL Studio tillverkas av Distributed Simulated Technology Inc. I GL Studio kan man importera eller rita grafiska figurer, för att sedan programmera lämpliga beteenden och egenskaper för dessa. I detta arbete har version 2.0 av programmet använts. För att kunna kompilera koden som genereras i detta program måste ett annat program användas. Till detta har Microsoft Visual Studio använts. 3ds max används för att skapa avancerade animationer och används bland annat i spelutveckling. Programmet är intressant i detta sammanhang då det ger möjligheter att visualisera flygsystem. Version 4.2 användes i detta arbete.

Det är många olika användare från olika miljöer som är tänkta att kunna skapa funktionsscheman. Dessa är programmerare och experter inom dataområdet, och ämnesexperter inom flygsystem. Detta arbete riktar in sig på en av dessa användare: programmerarna. I alla tre programmen krävs viss programmering för att skapa ett dynamiskt funktionsschema. Frågan är hur väl dessa användare kan använda sina programmerings-kunskaper när de skapar dynamiska scheman.

3 Problemformulering

Framgången har varit liten när det gäller att kategorisera in användare i grupper som i sin tur kan användas för att förutse en trolig respons på ny teknologi (Dillon & Watson, 1996). Man kan vara säker på att det finns skillnader mellan experter och noviser, eller att övning på ett system kommer att ge en ökad effektivitet. Däremot så misslyckas MDI–området enligt Dillon och Watson (1996) att indikera i förväg hur skillnader hos användarna relaterar till gränssnittsdesign, eller vilken sorts system som bäst skulle passa en specifik användargrupp. Det finns ett behov av forskning inom detta område (Dillon & Watson, 1996).

Den användargrupp som kommer att undersökas i denna studie är expertprogrammerare, och undersökningen kommer att rikta in sig på hur väl gränssnitt är anpassade efter deras speciella egenskaper. Enligt McKiethen et al. samt (1981) Soloway et al. (1988) använder expertprogrammerare sig av kognitiva scheman när de programmerar. Soloway et al. s (1988) studier inom programmering visar att experter har svårare att förstå ett program som inte följer dess invanda konventioner än de program som följer dessa. Noviser däremot, har lägre kunskap inom programmering och har därför inga invanda konventioner. Deras prestation skiljer sig därför inte lika mycket åt mellan de båda betingelserna. Det verkar alltså vara viktigt att experterna ska kunna använda sina kognitiva scheman för att prestera så bra som möjligt.

Enligt ISO-definitionen av användbarhet (se kap 2.3.1) ska användarna inte bara kunna lösa den tänkta uppgiften i ett system, de ska även kunna göra det så effektivt som möjligt. Om systemet inte stödjer expertens kunskap, antas det att experten inte kommer att kunna dra nytta av sitt kunnande. Hans prestation kommer då att bli den samma som hos en novis, detta i enlighet med studier inom schack (Chase & Simon, 1973) samt programmering (McKeithen, et al., 1981). Systemet är då inte effektivt, och har därför en låg grad av användbarhet.

Hur ska då ett system vara utformat för att vara anpassat för experter? Detta beror, enligt Gulliksen och Sandblad (1996) på vilken domän samt vilka experter det handlar om. För att kunna studera detta problem måste vi därför studera en specifik domän samt en specifik grupp experter. Därför kommer denna studie undersöka experter inom domänen programmering. Denna domän har förekommit i tidigare studier av expertis (McKeithen, et al., 1981; Soloway, et al., 1988), och därav kan man dra slutsatsen att det är en lämplig domän för att studera expertis, där experterna uppvisar liknande egenskaper.

3.1 Problemavgränsning

Denna studie avser testa hur ett system bör vara utformat för att ta tillvara på experters kunskap. För att studera detta har tre befintliga program valts. Genom att studera expertprogrammerarnas användning av dessa program kan man få information om huruvida användarna har problem med något i programmet, och om detta är fallet, med vad de har problem. De tre programmen är 3ds max, GL Studio samt VAPS. Studien kommer att inriktas på hur dessa program stödjer expertprogrammerarnas tidigare kunskap inom programmering, samt svara på frågan: Hur väl anpassade är programmen för sin användning och användargrupp? Då programmen är väldigt olika i sin utformning antas resultatet därför

kunna ge ledtrådar till hur ett program bör vara utformat för att passa experter. Dessa hypoteser har ställts upp:

- Då GL Studio till stor del är baserad på C++ kod, antas det att försökspersonerna kommer prestera bättre i detta program än i de övriga två.
- Då koden i 3ds max är speciell för det programmet, förväntas försökspersonerna prestera sämre i detta program än i de övriga programmen.

Dessa hypoteser baseras på antagandet att GL Studio, eftersom det är baserat på för programmerarna bekant programmeringsspråk, även kommer ha välbekanta begrepp som stämmer överens med programmerarnas kognitiva scheman. De kommer därför att prestera bäst i detta program. Med samma resonemang kommer inte försökspersonerna prestera sämre i 3ds max än i de övriga programmen, då programmeringen i detta verktyg sker med ett programmeringsspråk som inte är bekant för programmerarna. Begreppen i det programmet kommer då inte stämma med deras kognitiva scheman.

För att undersöka problemställningen kan man låta experter genomföra likvärdiga uppgifter i de tre programmen. Man kan sedan jämföra hur bra det gick för experterna att arbeta i programmet, med avseende på hur lång tid det tog att utföra uppgiften, hur många fel som gjordes samt vilka fel som gjordes. Resultatet förväntas ge ett bidrag till forskningsområdet genom att leda fram till nya frågor inom området, samt förslag på ytterligare studier.

4 Metod

För att avgöra hur användbara de tre aktuella programmen som ska utvärderas är, behöver en användbarhetstestning genomföras. Det finns många olika metoder för att få fram information om användbarheten hos en produkt. Dessa kan delas in i underkategorierna inspektionsmetoder, användartestning och intervjuer.

4.1 Inspektionsmetoder

Inspektionsmetoder innebär att en person eller en grupp av personer, oftast med kunskap om användbarhet, går igenom det aktuella systemet. Till inspektionsmetoder räknas bland annat heuristisk utvärdering och konceptuell genomgång.

4.1.1 Heuristisk utvärdering

Heuristisk utvärdering är en informell, subjektiv användbarhetstestning, utförd från den tänkta användarens synvinkel. Syftet är att belysa potentiella användbarhetsproblem, och därmed identifiera aspekter av gränssnittet som behöver förbättras för att öka användbarheten. Heuristisk utvärdering ger en snabb och grov uppfattning om möjliga problem. Både användbarhetsexperter och icke-experter kan utföra utvärderingen, men experter får bättre resultat (Karat, et al. 1992; Nielsen, 1992, i Lindgaard, 1994). En utvärdering går tillväga så att utvärderaren inspekterar gränssnittet för att sedan rapportera de problem som hittats antingen skriftligt eller muntligt. Det finns inga specifika metoder eller regler för hur utvärderaren ska gå tillväga, endast generella riktlinjer som ”tala användarens språk”. Därför beror resultatet på kunskapen hos utvärderaren (Lindgaard, 1994).

4.1.2 Konceptuell genomgång

I en konceptuell genomgång går en utvärderare igenom systemet för att upptäcka delar i systemet som är otydliga med avseende på användarens mål och handlingar. Utvärderaren väljer en uppgift att utföra i systemet, som sedan utförs. Utvärderaren går igenom de olika stegen i systemet som krävs för att användaren ska utföra sin uppgift, och undersöker gränssnittets beteende och effekt på användarna. Konceptuell genomgång identifierar olika diskrepanser som finns mellan systemets affordances och användarens mål (Lindgaard, 1994), det vill säga metoden undersöker hur väl systemet inbjuder till att utföra de handlingar som det är användarens mål att utföra.

Inspektionsmetoder skulle i detta fall ge svar på till vilken grad programmen är användbara, samt eventuellt vilka möjliga användbarhetsproblem programmen kunde ha, men valdes bort för att det ansågs viktigt att experterna själva är med i utvärderingen av systemet, då det är experternas mentala egenskaper i förhållande till gränssnittet som jag vill undersöka. Dessa egenskapers eventuella påverkan på användbarheten skulle inte gå att undersöka med inspektionsmetoder. Ett test där användarna, i det här fallet experterna, medverkar ansågs därför vara ett bättre alternativ.

4.2 Användartestning

Skillnaden mellan användartester och inspektionsmetoder är att i inspektionsmetoderna är det användbarhetsexperter som utför testerna, medan man i användartestning utför försök med de tänkta användarna. Detta görs genom olika experimentella metoder, och kan göras i ett laboratorium eller i verklig miljö beroende på vad man vill få för information av testet. Användartestning mäter användarens prestation i ett system. Denna sorts undersökning genomförs oftast som en observation. Observationer kan dock vara utformade på olika sätt.

4.2.1 Observation

Att observera användaren och föra protokoll över vad denna gör i systemet är mer pålitligt än att låta användaren förklara det samma (Sasse, 1991). Problemet med detta är att vi kan se vad användaren gör i systemet, men vi vet ingenting om varför. Metoden kan ge viktig information om vilka handlingar användaren utför i systemet, vilka typer av fel som förekommer samt ge möjligheten att upptäcka ineffektiva procedurer som användaren begagnar sig av. Observation är ett bra sätt att testa och utvärdera ett system, men bör kompletteras med någon form av undersökning av användarnas inre processer (Sasse, 1991). Detta kan ge den kompletterande informationen om varför användaren utför vissa handlingar i systemet. Ett sätt att göra detta är att låta användarna beskriva vad de gör i systemet, vanligtvis kallat ”tänka-högt-metoden”.

Tänka-högt-metodens tillvägagångssätt innebär att användarna verbaliserar sina tankar och slutsatser samtidigt som de interagerar med systemet. Användarna blir ombedda att bland annat förklara sitt eget beteende, resonera om systemets beteende, resonera om fel och lösa en uppgift verbalt innan de utför den i systemet. Ofta är syftet med interaktionen att lära eller utforska ett nytt system. Man spelar ofta in användarens arbete på video eller bandspelare, och använder sedan dessa inspelningar för att skapa skriftliga protokoll (Sasse, 1991).

Med hjälp av tänka-högt-metoden får man ofta fram mycket information, men det är inte alltid säkert att denna information är komplett. Sasse (1991) redogör för Normans (1983, i Sasse 1991) kritik mot metoden. Norman menar att människor ofta har svårt att verbalt redogöra för sina tankeprocesser. Att använda eller skapa en mental modell är ofta en omedveten process och kan därför inte verbaliseras. Om vi ändå ber användaren göra detta uppstår en onaturlig situation, och det kan hända att uppgiften att göra dessa omedvetna processer till medvetna förändrar de aktuella processerna. Informationen som fås blir då felaktig. Risken finns att användaren försöker rationalisera sitt beteende, eller säga det som de tror att försöksledaren vill höra. Enligt Hoffman et al. (1995) finns det dock forskning som visar motsatsen, det vill säga att verbalisering av tankeprocesserna inte påverkar tankeprocesserna. Han varnar däremot för att det kan finnas individuella skillnader i individers verbala uttrycksförmåga, vilket kan påverka resultatet av en undersökning.

Observation valdes till denna undersökning då denna metod hade flera fördelar. Observation tillåter användarna att delta i utvärderingsprocessen, något som ansågs vara mycket viktigt då det inte var användbarhet generellt hos systemen som skulle undersökas, utan hur försökspersonernas expertis påverkade användbarheten. En observation skulle även på ett bra sätt ge mätbara resultat. Genom att observera användaren kan man notera den tid det tar att

lösa uppgiften, hur många fel som användaren gör och dylikt. Dessa data skulle sedan kunna analyseras kvantitativt och ge svar på hypoteserna genom att svara på i vilket program som försökspersonerna presterade bäst. Ytterligare en fördel med observation ansågs vara att man genom observation inte bara kan få information om i hur försökspersonerna presterade i de olika programmen, utan även varför. Genom att observera användarna när de brukar de olika programmen kan man upptäcka vad det är i gränssnittet som användarna har problem med.

Då försöksledaren inte innehar samma kunskap om programmering som försökspersonerna valdes tänka-högt-metoden. Genom att låta försökspersonerna tänka högt kan de klargöra sina avsikter och på så sätt kan försöksledaren avgöra om den handling som observeras är ett fel eller ej. För att komplettera observationen valdes att också genomföra en intervju. Då användarens subjektiva åsikter är ett av kriterierna för användbarhet (Bevan 1997, se kap 2.3.1) ansågs det viktigt att dessa skulle registreras.

4.3 Intervjuer

Intervjuer är ett sätt att undersöka användarens kunskaper inom ett område, samt användarens subjektiva uppfattningar om ett system. Olika intervjuformer ger olika sorters information. Patel och Davidson (1994) anger grad av strukturering och grad av standardisering som ett sätt att avgöra vilka sorts svar man kan få fram. En låg grad av standardisering innebär att försöksledaren under intervjun själv formulerar frågorna och ställer frågorna i den ordning som passar intervjupersonen. Vid en hög grad av standardisering ställs samma frågor i samma ordning till samtliga intervjupersoner. Graden av struktur avgör hur öppna svarsalternativen är för en intervjuperson.

En ostrukturerad intervju antar ofta formen av en öppen dialog. Intervjuaren ställer frågor med öppna svarsalternativ. Ett exempel på detta är: ”Berätta allt du vet om X.” En intervju av detta slag kan ge en bra överblick över en domän. Intervjuaren kan ha flera olika intervjusessioner och på så sätt få en heltäckande bild av domänen. Det finns dock vissa nackdelar med denna metod. Till exempel kan användaren komma in på fel spår under intervjun, eller anta att försöksledaren har kunskap om området som den inte har. I övrigt så kan en intervju av ostrukturerad natur vara svårare än mer strukturerade att transkribera, en process som ändå tar mycket lång tid (Hoffman, et al., 1995).

En strukturerad intervju kan spara mycket tid jämfört med en ostrukturerad intervju. I denna intervjuform används oftast på förhand bestämda frågor, designade för att ge en inblick i problemområdet. Denna typ av intervjuform är bra när man vill bekräfta information man redan har samlat in (Hoffman, et al., 1995).

Trots den kritik mot öppna intervjuer som nämnts ovan valdes intervjun ändå att genomföras som en ostrukturerad intervju med öppna svarsalternativ. En öppen intervju ger möjligheten att föra ett samtal med försökspersonerna om vad som just hänt under observationen. En öppen intervju skulle ge möjligheten att förändra frågorna efter varje försökspersons individuella situation och på så sätt komplettera observationen samt även ge möjlighet till att reda ut eventuella missförstånd som skett under observationen.

4.4 Försökspersoner

Som försökspersoner användes personalen på AerotechTelub i Arboga. De arbetar alla som programmerare och har flera års erfarenhet inom området. De tillfrågades också om sin programmeringsvana under testet. Antalet försökspersoner var sex stycken, fem män och en kvinna. Medelåldern var 32 år. De hade arbetat med programmering i genomsnitt 8,4 år. Spridningen var dock stor. Den mest erfarne hade 16 års erfarenhet av programmering och den minst erfarna 2,5 år.

4.5 Upplägg och material

För att kunna observera experterna när de utförde en uppgift i respektive program utformades tester för att få reda på om försökspersonerna skulle göra fel i de olika programmen, och i så fall vad de skulle ha problem med. För att få reda på detta fick försökspersonerna lösa en uppgift i var och ett av programmen. En inomgruppsdesign användes, det vill säga alla försökspersoner fick arbeta i alla program. Denna uppläggning valdes för att balansera för individuella skillnader som kunde finnas mellan försökspersonerna avseende programmeringsskicklighet. Dock finns det risk för differential transfer när man använder inomgruppsdesign. Differential transfer innebär att resultatet av en betingelse påverkas av vilken betingelse som tidigare presenterats för försökspersonen (Shaughnessy & Zechmeister, 1990). Differential transfer kan exempelvis uppkomma om man ger försökspersonerna en viss instruktion i betingelse A, som dom sedan inte kan bortse från när de ska testas i betingelse B. Differential transfer kan varken kontrolleras eller balanseras, och föreligger risk för differential transfer bör man välja en annan experimentell design (Shaughnessy & Zechmeister 1990). I denna undersökning förväntades det dock inte bli några problem med differential transfer då inget program förväntades påverka arbetet i något av de andra på ett systematiskt sätt. Däremot fanns en risk för övningseffekter, det vill säga att försökspersonerna skulle lära sig olika lösningar från ett program till ett annat, eller använda lösningar de använt i ett tidigare program och därför göra fel. För att sprida ut påverkan från eventuella övningseffekter på resultatet användes ”alla möjliga ordningar” för att balansera för detta.

Uppgifterna gick ut på att lösa ett problem i vart och ett av programmen. Men efter ett förtest av uppgifterna innan testet upptäcktes att dessa uppgifter var alltför komplexa, då försökspersonerna aldrig jobbat i programmen förut. Därför gjordes en del ändringar: Innan de skulle lösa uppgiften fick de en kort introduktion i programmets gränssnitt för att kunna orientera sig bättre. Annars skulle försöken ta alldeles för lång tid, då det under förtestet tog flera timmar att lösa ett enkelt problem. Uppgifterna ändrades till att rätta felen i en kod som redan fanns färdig i programmet därför att annars skulle uppgifternas ta alltför lång tid. När försökspersonen gjort rättningar skulle programmet fungera. För att kunna jämföra de olika programmen med varandra fanns en önskan om att utforma uppgifterna i de tre programmen lika. Detta var dock inte möjligt på grund av programmens olika natur. En syssla som inte krävde någon programmering för att utföras i ett program krävde mycket programmering för att utföras i ett annat, och vice versa. De utformades däremot med hjälp av en programmerare för att de skulle bli så likvärdiga i fråga om svårighetsgrad som möjligt. Om uppgifterna var lika svåra programmeringsmässigt, skulle de eventuella skillnader som framkom bero inte på hur svår uppgiften var, utan på hur svår den var att utföra i just det programmet. Eventuella skillnader mellan de olika programmen skulle då bero på programmens användbarhet.

De slutliga uppgifterna såg ut enligt följande: I GL Studio skulle användaren rotera en pil i en mätare. Pilen och mätaren fanns redan skapade, försökspersonen skulle definiera en funktion som styrde objektets rörelser. Till användarens hjälp fanns fördefinierade funktioner som kunde användas. För att lösa uppgiften behövde användaren välja två av dessa funktioner: ”Mouse_drag” samt ”dynamic rotate”. När den gjort detta behövde användaren specificera vilket objekt som skulle roteras (det vill säga namnet på objektet), samt hur många grader objektet skulle rotera. Totalt krävdes fyra rader kod för att slutföra uppgiften.

Uppgiften i VAPS innebar att försökspersonerna skulle rätta felaktig kod så att en lampa, i form av en pil, lös när de tryckte på en knapp. Innan de började med uppgiften fick de se hur uppgiften skulle se ut när de var klara. I VAPS fick försökspersonerna se hur koden såg ut sammansatt, i ett anteckningar-fönster. Anledningen till detta var att uppgiften inte skulle bli allt för svår. I koden var två variabelnamn utbytta mot *-tecken. Variabelnamnen var namnet på knappen, ett default-namn programmet givit knappen automatiskt. Namnet var ”button 2”. Totalt krävdes alltså två ändringar för att rätta koden.

I 3ds max gick uppgiften ut på att rätta ett felaktigt script. När scriptet var komplett skulle det skapa en knapp samt tre radiobuttons i gränssnittet. När man klickade på knappen skulle en låda skapas av den färg som var vald i radiobuttons. I scriptet fanns fyra fel: Variabelnamnet på radiobuttons, samt att de hade ett tillstånd, ”.state”, hade tagits bort på tre ställen och ersatts med ”XXXX”. Utöver detta hade även ett tilldelningsfel smugits in i koden, då ett ”= =” hade ändrats till ett ”=”. Totalt behövde försökspersonen ändra på fyra ställen för att klara uppgiften.

Som beroende variabler valdes att mäta hur lång tid det tog för försökspersonerna att klara uppgifterna, hur många omkompileringar som försökspersonerna gjorde samt hur många gånger försökspersonerna tittade i hjälpfilen. Dessa anses vara bra kvantitativa data på hur väl produkten går att använda (Dumas & Redish, 1993). Tid valdes som beroende variabel därför att det antogs vara ett bra mått på försökspersonens prestation i programmet. Ju mer problem försökspersonen hade i programmet, desto längre tid antogs det skulle passera innan uppgiften var löst. Det visade sig vara väldigt svårt att definiera vad som kunde räknas som ett fel av användaren. Då försökspersonerna skulle vara tvungna att utforska programmen för att lösa uppgifterna gick det inte att mäta feltryckningar och felnavigering som fel. Istället valdes att mäta antalet kompileringar som försökspersonen gjorde. Det antogs att desto mer fel försökspersonen gjorde, desto fler gånger skulle försökspersonerna vara tvungna att kompilera om programmet för att få det att fungera korrekt. För att avgöra hur väl försökspersonerna förstod programmet mättes hur många gånger de tittade i programmets hjälpfiler. Det antogs att ju svårare försökspersonen har för att förstå programmet, desto fler gånger måste försökspersonen läsa i hjälpfilen för att lösa uppgiften.

För att försöken inte skulle ta alltför lång tid sattes en tidsgräns på uppgifterna. Det bestämdes att dessa inte fick ta mer än trettio minuter, då skulle försökspersonen sluta med den uppgiften. Detta gjordes för att begränsa den totala tiden som försöket skulle ta, då många av försökspersonerna hade ont om tid för att deltaga. Trettio minuter verkade som en rimlig tid då försökspersonen på förtestet klarat uppgifterna på maximalt 25 minuter. Då uppgifterna nu blivit lättare, så antogs trettio minuter vara tillräckligt med tid. Med trettio minuter per program planerades undersökningen ta cirka 2 timmar för varje försöksperson.

4.6 Genomförande

Innan försöket började hälsades den deltagande välkommen, och fick sedan information om hur försöket skulle gå till. De informerades om tänka-högt-metoden som skulle användas, och hur detta skulle gå till. De fick även information om att de under hela försöket hade rätt att avbryta om de kände någon form av obehag. De informerades också om att det var programmen som skulle testas och ej försökspersonernas individuella prestation. Detta gjordes för att försökspersonerna inte skulle känna någon onödig press. De försäkrades om att materialet bara skulle användas till denna undersökning samt att resultatet var anonymt. För att alla försökspersoner skulle få samma information och att ingen viktig information skulle glömmas, gavs denna information på papper till försökspersonerna (se bilaga 1).

Under försöket mättes hur lång tid varje uppgift tog, antal omkompileringar, samt antal gånger som de behövde läsa i hjälpfilen. Observationerna spelades även in på video för att lättare kunna analyseras senare.

När försökspersonerna hade gjort alla tre uppgifterna avslutades försöket med en intervju. Här tillfrågades försökspersonerna om hur de tyckt det varit att arbeta i programmen, vad de haft problem med och liknande. För att ge stöd för intervjuaren fanns det några frågor nedskrivna (se bilaga 2). Först frågades försökspersonerna om deras ålder och programmeringsvana. Dessa frågor gav viktig information, men fungerade även som en introduktion till intervjusituationen, då de inte var särskilt krävande men satte igång konversationen. Efter dessa frågor fick försökspersonen berätta om vilka program han eller hon brukade arbeta med. Till sist gick intervjun in på vad som hänt under försöket och vad försökspersonen ansett om att arbeta i de olika programmen. Intervjuerna spelades in på video för att kunna transkriberas vid ett senare tillfälle.

Under försöket framkom att den satta tidsgränsen på trettio minuter per uppgift inte var tillräcklig. Endast hälften av uppgifterna klarades inom den utsatta tiden. Under försöket fanns också vissa problem med ett av programmen, VAPS. Detta program krånglade under ett av försöken. Den tid det tog att starta om datorn räknades dock bort från den tid det tog för försökspersonen att lösa uppgiften. Ett annat problem var att de olika programmen inte kunde köras på samma dator. Försökspersonerna fick därför byta kontor då de skulle utför uppgiften i VAPS. Kontoren var likvärdiga till utseende, men förflyttningen kan ha stört försökspersonerna något. Ingen påpekade dock detta som ett störande moment, vilket tas som ett tecken på att störningen var minimal.

5 Resultat och diskussion

Undersökningen innehöll både kvantitativa samt kvalitativa inslag. Dessa kommer att presenteras var för sig, för att sedan sammanfattas på slutet.

5.1 Kvantitativ analys

Syftet med undersökningen var att utforska området gränssnittsdesign för domänexperter. Då GL Studio använde sig av programmeringsspråket C++, som är familjärt för programmerarna antogs att de begrepp som används i gränssnittet är bekanta. Därför antogs att GL Studio skulle vara bättre i användningen. Användarna förväntades ta kortare tid i anspråk för att utföra uppgiften, behöva färre kompileringar samt behöva läsa färre antal i hjälpfilen i det programmet än i de andra. Då 3ds max använder ett eget språk, MaxScript, antogs att de begrepp som användes inte skulle passa med programmerarnas kognitiva scheman. Uppgiften i detta program förväntades därför gå sämre. Här förväntades att uppgiften skulle ta längre tid, kräva fler kompileringar samt läsning i hjälpfilen fler gånger.

5.1.1 Resultat

Tabell 1 visar de olika medelvärdena samt standardavvikelserna för de olika programmen samt de olika beroende variablerna.

Tabell 1: Översikt över medelvärden och standardavvikelser.

	GL Studio	3ds max	VAPS
Tid	**26,50** (SD=6,12)	**18,50** (SD=10,31)	**23,83** (SD=9,55)
Antal kompileringar	**3,17** (SD=1,94)	**5,00** (SD=2,44)	**2,17** (SD=2,04)
Antal anv. hjälpfil	**3,17** (SD=1,47)	**1,50** (SD=0,84)	**2,00** (SD=0,89)

Tid

Enligt medelvärdena var det uppgiften i GL Studio som tog längst tid att utföra: (M=26,50). Uppgiften i VAPS tog näst längst tid att utföra (M=23,89), medan uppgiften i 3ds max tog kortast tid att slutföra (M=18,50). Skillnaden mellan värdena beräknades med ANOVA. Skillnaden var dock inte signifikant, $F(2,10)=2,009$, MSe=49,556, $p=0,185$. Då endast 8 av de 18 uppgifterna klarades inom den angivna tidsramen 30 minuter, finns här risk för att eventuella skillnader mellan programmen inte syns. Därför gjordes ett χ^2 -test för att räkna ut om det var någon av uppgifterna i de olika programmen som slutfördes oftare än de andra. Två av sex personer hade slutfört sin uppgift i GL Studio. För 3ds max respektive VAPS var antalet slutförda uppgifter fyra samt två. Resultatet var dock inte signifikant: $\chi^2(2) = 1,800$, $p=0,407$

Kompilering

Medelvärdena visar att VAPS kompilerades minst antal gånger (M=2,17). GL Studio kompilerades något fler gånger (M=3,17). 3ds max kompilerades flest gånger (M=5,00). En variansanalys visade att dessa skillnader var signifikanta, $F(2,10)=6,027$, MSe=2,056, $p=0,019$. För att avgöra var skillnaden fanns gjordes planerade analytiska jämförelser mellan de olika programmen. Dessa tester visade att VAPS skilde sig från de andra programmen, $F(1,5)=10,292$, MSe=8,567, $p=0,024$. GL Studio skiljde sig däremot inte från de andra programmen, $F(1,5)=0,566$, MSe=7,367 $p=0,468$. Inte heller 3ds max skilde sig från de andra programmen, $F(1,5)=6,203$, MSe=21,067, $p=0,055$. VAPS kompilerades alltså signifikant färre gånger än de andra programmen.

Användning av hjälpfilen

Medelvärdena för användning av hjälpfilen var följande: GL Studio (M=3,17), 3ds max (M=1,50), samt VAPS (M=2,00). Skillnaden mellan dessa var signifikant, $F(2,10)=4,158$, MSe=1,056, $p=0,049$. Även här gjordes planerade analytiska jämförelser för att avgöra var skillnaden fanns. De olika programmen jämfördes var för sig mot de andra två. Inget resultat var dock signifikant enligt följande uträkningar: GL Studio jämfört med de andra programmen: $F(1,5)=5,142$, MSe=9,367, $p=0,073$. 3ds max jämfört med de övriga två programmen: $F(1,5)=3,288$, MSe=8,567, $p=0,130$. VAPS jämfört med de övriga två programmen: $F(1,5)=2,500$, MSe=1,067, $p=0,175$.

5.1.2 Diskussion

Studien undersökte hur ett program ska vara utformat för att stödja expertprogrammerares kunskap. Hypoteserna var att försökspersonerna skulle få ett bättre resultat i GL Studio än i de andra programmen eftersom det är baserat på ett för programmerarna välbekant programmeringsspråk, samt att 3ds max *inte* skulle få ett bättre resultat än de andra eftersom det inte använder ett för programmerarna bekant programmeringsspråk. Detta kunde inte visas i den kvantitativa analysen. Ingen skillnad upptäcktes mellan 3ds max och de andra programmen, oavsett beroende variabel. Ingen skillnad fanns heller mellan GL Studio och de andra programmen. Detta resultat kan bero på brister i metoden. En brist är tidsgränsen som fanns för uppgifterna i försöket. Då bara 8 av de 18 uppgifterna lyckades slutföras inom tidsgränsen, uppstod en takeffekt. Detta innebär att undersökningen inte var tillräckligt känslig för att finna en skillnad mellan de olika programmen. Om försökspersonerna tillåtits att arbeta tills de hade löst uppgiften hade en sådan effekt kunnat undvikas.

VAPS hade signifikant färre omkompileringar än de andra programmen. Detta resultat kan dock inte tolkas som att VAPS var lättare att använda, och därför inte behövdes omkompileras så mycket. Tvärt om, så hade försökspersonerna problem med att kompilera i programmet. Fem av försökspersonerna observerades ha problem med detta (se kapitel 5.2.2.3). Att VAPS har så få kompileringar beror snarare på att försökspersonerna hade svårigheter att utföra kompileringen i det programmet.

5.2 Kvalitativ analys

5.2.1 Analys av intervjuerna

Analysen av intervjuerna har gjorts genom att studera materialet och tolka det som hände i undersökningen. Tolkningen har gjorts utifrån de uttalanden som försökspersonerna gjort samt händelserna under observationen, med grund i de teorier som presenterades i kapitel 2. Försökspersonernas uttalanden kommer att presenteras som citat. På en del ställen har förtydligande kommentarer behövts, då citaten är tagna ur ett större sammanhang. Dessa kommentarer har markerats med hakparenteser. I detta kapitel tas endast vissa av intervjusvaren upp (för kompletta svar se bilaga 3, där de fullständiga intervjuerna finns).

Enligt hypoteserna i kapitel 3 skulle försökspersonerna finna det lättast att jobba i GL Studio, då detta program använde sig av programmeringsspråket C++, vilket de är vana vid. De skulle, enligt samma tankesätt ha svårt att använda sig av 3ds max då detta program använde en egen variant av objektorienterad kod kallad MaxScript. Antagandet var att de kognitiva scheman som expertprogrammerarna har inte skulle kunna användas till lika hög grad i 3ds max. Resultatet av undersökningen är tvetydigt i denna fråga. Några av försökspersonerna ansåg att GL Studio var det bästa programmet:

> ”Och det här GL Studio tyckte jag var bäst, men där var hjälpen dålig. Den var verkligen i sämsta laget.” (Intervju 3)

> ”Jag tyckte nog GL var bäst, det var enklast, enklast att använda på nåt sätt. Det kändes mer lika C++ också, det som jag använt.” (Intervju 2)

> ”Det var väl den som jag kände mig mest hemma i, men det kan ju ha att göra med att det var kopplat till Visual Studio, och att det var C++.” (Intervju 1)

Försökspersonerna har uttryckt att de känner sig mest hemma i GL Studio. Detta kan bero på att GL Studio använder ett för dem bekant programmeringsspråk. Att programmeringsspråket är det samma som de är vana att arbeta i kan göra så att försökspersonerna kan använda sina kognitiva scheman när de använder detta program. Detta kan dock vara svårt att visa, då de tre personer som angav GL Studio som det mest användbara programmet alla har vana av att arbeta i Visual Studio sedan förut. I den kvantitativa analysen fick GL Studio inte heller något bättre resultat än de andra programmen. Flera personer uttryckte också problem med hur programmet var utformat:

> ”I: Vad tyckte du att du hade svårigheter med i GL Studio?
>
> FP: Svårigheter var att, det som rörde till sig för mig, det var ju att jag tittade i Visual Studio så såg jag mer kod än vad jag såg här. Då vart jag osäker. Kan jag använda den här koden? Finns det någon vits med att jag inte ska se den? Antagligen så finns det väl det, men ändå så ville jag ju köra med den. Hade jag inte tittat där så kanske det hade gått mycket bättre. Det var liksom delar av koden som man inte visste vart de hörde.” (Intervju 4)

> ”Ja, det var likadant som det här [VAPS], det var inte uppenbart hur man skulle göra.” (Intervju 5)

I båda citaten antyds att det var svårt att förstå vad som skulle göras utan att få en ordentlig överblick över koden. I både GL Studio och VAPS används separata fönster för att skriva den funktion som styr objektens rörelser. Den fullständiga koden är inte synlig för programmeraren. Detta verkar inte passa programmerarnas arbetssätt, då tre av personerna uttryckte att de helst ville ha en överblick över koden då de arbetar:

> ”I jämförelse med den här, här har dom ju försökt att i viss mån gömma bort sammanhanget för programmeringen innan, ja, man kunde ju markera en hel rad där men… ja det känns som dom har misslyckats med att gömma rätt saker.” (Intervju 5)

> ”Ja, rent generellt så, vilka arbetsmoment du ska göra, hur det hänger ihop. Sammanhanget. Ja menar, varje grej kan man ju liksom förstå att, ja det här är ju nån tabell över en statement, det har man ju liksom läst om och använt i olika projekt och så vidare, men hur hänger det ihop då med det övriga? Det ser man inte, det är precis som om dom har lagt ett skynke över vissa bitar, så man liksom inte kan titta efter hur det funkar och så där.” (Intervju 5)

> ”Jag är mer van vid att sitta och bara programmera, skriva kod rakt så rent av. Det känns som om man vet mer vad som händer än när man sitter och…, det är nog mycket lättare också när man har lärt sig ett sådant här verktyg helt så man vet vad man gör. Men när man bara sitter här så är det mycket svårare och det tog mycket längre tid, än att bara sitta med koden.” (Intervju 2)

Dessa citat indikerar att gränssnittet var ett hinder för själva programmeringen, både i VAPS och i GL Studio. De funktioner i programmen som till synes är gjorda för att underlätta för programmerarna var i stället ett hinder i deras arbete. Att begränsa tillgängligheten för koden var alltså försvårande i stället för förenklande. En försöksperson beskrev det som ”att gå bakvägen” för att lista ut hur programmet skulle användas:

> ”Ja, jag tror det var Visual Studio, att man kan få fram allting i C-kod. Då ser man verkligen vad som händer och då kan man lättare lista ut hur man ska använda verktyget. Så det blir ju som att gå bakvägen för att lista ut hur man ska använda ett förenklat verktyg. Det kanske är fel sätt att använda det på.” (Intervju 3)

En annan indikering på att det underlättar för expertprogrammerare att få en överblick över hela koden var att många lyckades bra på uppgiften i 3ds max. Denna uppgift var det flest personer som klarade, till skillnad från vad som antagits i hypoteserna. En av förklaringarna

till detta kan vara det enkla gränssnittet. Två försökspersoner uttryckte att de tyckte att 3ds max var det enklaste programmet:

> ”FP: Det är klart, kan man scriptspråket så, då är det väl inga större problem. Så då tror jag att det här var det mest lättförståeliga, det bästa av dom tre.
>
> I: Varför tycker du så?
>
> FP: Ja, men liksom, man fick upp koden på ett ställe och så bara skrev man i den. Det var inte så mycket hoppa mellan olika fönster kors och tvärs. Och framförallt inte mellan olika program.” (Intervju 4)

> ”Jag tror nog 3ds max var lättast att komma in i. Jag lyckades ju faktiskt lösa uppgiften, så att… eh… så det kändes ju mer, man förstod intuitivt vilket grepp man skulle göra, om man säger så.” (Intervju 5)

Även andra försökspersoner verkade tycka att 3ds max var enkelt att använda trots att det använde sig av en annorlunda syntax än de var vana vid, beroende på det okomplicerade gränssnittet:

> ”Men ja sen, programmeringen i 3ds den var ju helt okej. Nu var det ju scripts och det kändes lite mer…, ja det var ju lite mer HTML-inriktat och det är jag ju inte så van vid, men det var ju enkelt, då skriver man bara koden och testar om den funkar, så det är nog bra.” (Intervju 2)

> ”Och 3ds max, de var väl bättre, jag såg ju bara ett skrap på ytan, vad jag kände dårå, men det var inte helt svårt, det var bara att tuffa på och komma på hur man skulle göra.” (Intervju 3)

Intressant att fråga sig här är om en novis skulle ha samma resultat, och finna de program där de kunde arbeta direkt med den rena koden lättare att arbeta med än de program där det fanns färdiga kodstrukturer i programmet, och användaren inte kunde se hela programmkoden. Detta är svårt att avgöra, då inga noviser fanns med i min undersökning som jämförelse. För att kunna avgöra detta krävs ytterligare studier.

En annan förklaring till att expertprogrammerarna föredrar att se hela koden kan vara att de är vana att arbeta i sådana utvecklingsmiljöer. Att utesluta denna förklaring är omöjligt, då alla expertprogrammerare har vana från olika utvecklingsmiljöer och program. Detta kan göra att programmerarnas kognitiva scheman är beroende av vilket program de är vana vid.

Man kan inte heller avfärda att förklaringen till 3ds max’s resultat kan vara att den uppgiften var enklare än de andra, vilket gjorde att försökspersonerna presterade bättre och blev mer positivt inställda till programmet. Även om uppgifterna utformades för att ha likvärdig svårighetsgrad, så var detta mycket svårt då det är svårt att skilja svårigheter som beror på

programmeringen från svårigheter som beror på gränssnittet. Programmeringen och gränssnitten hänger samman och är svåra att behandla separat.

Andra aspekter som kan ha påverkat resultatet är de begrepp som användes i gränssnittet. Två av försökspersonerna hade problem när de använde programmet VAPS, då de inte kände igen de begrepp och defaultvärden som programmet gett:

> ”VAPS tyckte jag verkligen skilde sig för det var svårt att förstå vad dom menade med de olika variablerna…” (Intervju 2)

> ”FP: VAPS, ja… Jag vet inte, det är säkert bra när man lär sig, men helt intuitivt tyckte jag inte att det var.
>
> I: Varför inte?
>
> FP: Det beror nog främst på att man inte kände igen begreppen.” (Intervju 4)

Detta kan anses vara ett stort användbarhetsproblem för experterna, då de begrepp som används inte stämmer överens med deras kognitiva scheman. En annan förklaring kan vara att det är de begrepp som hör samman med de grafiska funktionerna i programmen som var svåra att förstå. Alla tre programmen innehöll behandling av grafiska objekt, men endast VAPS fick dessa kommentarer. Däremot framkom vissa andra problem med andra programmen under observationen (se kap 5.2.2.1).

5.2.2 Analys av observationerna

Här kommer att presenteras vad som hände under observationerna. Observationerna finns ej transkriberade då kommentarerna under observationerna hör samman och behandlar det som händer på försökspersonens dataskärm, och därför är väldigt svåra att förstå fristående.

5.2.2.1 Gränssnitten hinder för programmerarna?

Under observationen fanns tecken på att programmens gränssnitt var hinder för programmerarna då de programmerade:

Under arbetet i GL Studio uttryckte två försökspersoner problem då det inte går att se hela koden i själva GL Studio, utan att ett ytterligare program (Visual Studio) behövs. En av dem uttryckte det som att det inte fanns någon ”kontext” till koden.

I VAPS använde endast en av försökspersonerna de rutor som finns där man kan klicka fram koden genom att välja olika händelser och variabler, alla övriga skrev den för hand.

Flera personer hade problem med VAPS gränssnitt: Tre av försökspersonerna uttryckte under arbetets gång att de ”inte alls visste vad de skulle göra” samt ”det var inte helt intuitivt hur man skulle göra”. En person sa att han inte såg några samband alls mellan de olika fönstren i programmet.

Resultatet här pekar på samma sak som försökspersonerna nämnde under intervjun, nämligen att överblick över koden är viktigt. Det verkar som om försökspersonerna föredrar att skriva koden själva snarare än att använda de hjälpmedel som finns i programmen. Dessa hjälpmedel verkar snarare vara ett hinder än en hjälp. Detta kan bero på att försökspersonerna inte kan använda sina kognitiva scheman när de programmerar, utan måste använda ett nytt tankesätt när de använder de hjälpmedel som finns i programmen. De måste då lära sig hjälpmedlen från början, och kan inte dra nytta av sin tidigare kunskap. Vilka begrepp som används i programmen kan också ha betydelse. Under observationen framkom flera fall där användarna hade svårt att förstå de begrepp som användes i gränssnittet:

I uppgiften i VAPS observerades följande problem: Tre personer hade problem att förstå att default-värdena ”code 1” och ”code 2” är variabelnamn. De antog att det var någon sorts kod som skulle köras vid det anropet.

I uppgiften som utfördes i 3ds max hade fyra av försökspersonerna problem med att lägga till ändelsen ”.state” till radiobuttons-objekten. En person anmärkte även att det var fel att ha med ett ”ä” i variabelnamnet.

Under utförandet av uppgiften i GL Studio hade tre av försökspersonerna problem med att i koden ange namnet på det objekt som skulle roteras. De använde ”input-pil” istället för ”pil”.

Dessa är tydliga exempel på situationer där arbetet i programmen skiljer sig från de programmeringskonventioner som programmerarna är vana vid. Begreppen som används av programmen är inte bekanta för programmerarna, vilket gör att de inte kan använda sin tidigare kunskap för att förstå vad dessa innebär. Exemplen ovan visar att det är viktigt att använda begrepp och default-värden som stämmer överens med de begrepp som används av programmerarna. Däremot är det inte rimligt att känna till och förstå alla begrepp i ett helt nytt gränssnitt. Eftersom programmen behandlar grafik finns det många begrepp som är förknippade med detta område som är nya för försökspersonerna. Detta kan vara en orsak till att försökspersonerna hade problem med att tolka begreppen..

5.2.2.2 Felhantering

Försökspersonerna hade också svårigheter med att hantera de fel som uppkom:

I uppgiften i GL Studio uppkom följande problem:
En person hade skrivit fel i koden, och kunde därför inte få pilen att röra sig, men kunde ej upptäcka problemet. Personen började då leta efter andra fel, och ansåg att felet var att rörelsen i pilen inte ritades upp på skärmen.

Tre av försökspersoner hade problem med de färdiga funktioner som fanns att tillgå för att lösa uppgiften. Alla tre tyckte inte att den kod som genererades av dessa funktioner var tillräcklig för att lösa uppgiften, och började därför skriva in ny, för uppgiften ej nödvändig, kod.

I VAPS uppkom följande situationer:
Tre av personerna lyckades rätta koden korrekt, men inte kompilera denna och se att den var rätt. De började därför söka efter fel i koden, och ändrade i koden, trots att den var rätt.

En person kompilerade koden på fel sätt, och fick därför konstiga felmeddelanden som gjorde att personen ändrade den korrekta koden, och började skapa filer som inte var nödvändiga för att lösa uppgiften.

I 3ds max fanns följande problem:
En person frångick koden och började söka i resten av programmet när den inte lyckades lösa problemet med hjälp av kod.

Två personer hade svårt att hitta tilldelningsfelet som fanns i koden. Båda började ändra på andra ställen där koden var korrekt, när de inte kunde tolka de felmeddelanden som de fick.

Experter är bra på att hantera fel inom sin domän därför att de redan genom sina interna scheman byggt upp lämplig respons på ett problem (Glaser & Chi, 1988). De problem som redovisas ovan kan bero på att försökspersonernas scheman inte stämmer, då problemen de identifierat inte kan lösas på de sätt dom är vana vid. Deras scheman för det aktuella problemet leder då till en felaktig lösning.

En annan orsak till problemen kan vara att de inte lyckas identifiera problemen på ett riktigt sätt, och därför passar inte den respons på problemet som fanns i deras scheman. Det första exemplet från GL Studio tycks styrka denna uppfattning då personen tillämpar en felaktig förklaringsmodell till att pilen inte rör sig; rörelsen ritas inte upp på skärmen. Detta antyder att personen kategoriserat felet fel, och därför fungerar inte hans lösning på problemet.

5.2.2.3 Övriga händelser

Under observationen uppkom även fler problem.

I GL Studio-uppgiften iakttogs följande problem:

Fyra av försökspersonerna uttryckte under observationen problem med hjälpen som finns i programmet. Den är gjord i pdf-format, vilket gjorde att försökspersonerna hade svårt att söka efter den information som de behövde.

En person hade svårigheter med att det krävs att först godkänna förändringar i koden, sedan spara programfilen, sedan generera nya C++ filer, som man sedan fick kompilera och exekvera via Visual Studio. Personen glömde ofta något av de olika stegen, och fann detta mycket frustrerande.

Under VAPS uppgiften hände följande:
Fem av försökspersonerna hade problem med att kompilera och köra programmet.

Tre av personerna hade problem att hitta det speciella program där man kör det färdigkompilerade programmet.

Två personer hade svårt att hitta hjälpfilen.

En person klagade på dålig koppling mellan felmeddelandet och rutan med koden.

I 3ds max observerades följande problem:

Två personer tyckte det var omständigt att hela tiden spara scriptet för att sedan ladda in det igen när det skulle testas.

Dessa problem är relaterade till användbarheten hos programmet, men kan inte sägas vara specifika för experter. Dock kan användbarhetsproblemen hindra programmerarna från att använda sitt vanliga arbetssätt. Problemen med att kompilera kan exempelvis vara problematiska då en programmerare vill kunna kompilera om ofta för att kontrollera sin kod. Detta framkom även under intervjun:

> ”När man sitter och, när man skriver, ja man kompilerar ju ganska ofta, bara för att testa om det lilla man har skrivit funkar, så ska man göra det där var tredje sekund eller vad det nu blir så blir det ju rätt så mycket.” (Intervju 4)

Att försökspersonerna hade svårt att hitta i programmen kan bero på att programmens uppbyggnad inte följde en för experterna naturlig struktur. Detta kan dock inte avgöras i denna undersökning, då det inte går att jämföra resultatet med hur personer som inte har experternas erfarenhet skulle ha presterat.

5.3 Sammanfattning av resultatet

Resultaten från den kvantitativa analysen visar att det inte finns något stöd för hypoteserna att GL Studio skulle vara bättre i användningen än de andra programmen var. Den visar även att den andra hypotesen, att 3ds max skulle vara sämre i användningen än de andra, är felaktig. Här visar dessutom den kvalitativa analysen att många av försökspersonerna faktiskt föredrog 3ds max. Förklaringen till detta kan vara att det sättet programmet är uppbyggt på, passade försökspersonernas arbetssätt. 3ds max har en kod-editor där man skriver koden rakt upp och ned, något som verkade uppskattas av försökspersonerna. En alternativ förklaring till 3ds max:s resultat kan vara hur uppgiften var utformad. Om denna uppgift var lättare att utföra än de övriga skulle detta höja försökspersonernas prestation, samt även tack vare det göra dem mer positiva till gränssnittet.

En del försökspersoner föredrog att arbeta i GL Studio och nämnde som orsaker till detta att programmeringsspråket C++ samt programmet som användes som kompilator (Visual Studio) var bekanta för dem. Användningen av Visual Studio kan vara en trolig förklaring till att GL Studio fick bra omdömen under intervjun.

Programmet VAPS var enligt försökspersonerna svårt att använda. Avsaknaden av överblick över koden kan vara en av anledningarna till detta, användningen av för programmerarna ofamiljära begrepp kan vara en annan. VAPS hade i den kvantitativa analysen signifikant färre antal kompileringar än de andra två programmen. Detta berodde dock förmodligen inte på att programmet var så lätta att använda att de löste uppgiften utan många fel. Detta resultat kan snarare förklaras med hjälp av den kvalitativa analysen, där det framkom att många av försökspersonerna hade det svårt att i gränssnittet hitta hur man skulle kompilera i programmet, och hur man skulle testköra det program man rättat.

5.3.1 Metodkritiska synpunkter

Under undersökningen förekom vissa svårigheter. Under observationen var det väldigt svårt att få försökspersonerna att använda tänka-högt metoden. Då de började fundera på ett problem blev de tysta. För att få dem att berätta vad de tänkte på tillfrågades dem efter en stunds tystnad vad de tänkte på. Detta kan ha stört deras arbete och därför påverkat resultatet. Deras tystnad är dock allvarligare. När försökspersonerna inte redogör för sina tankar och handlingar kan deras beteende misstolkas, vilket kan leda till felaktiga slutsatser. Dock kan resultatet från intervjun motverka detta eftersom försökspersonerna där fick chansen att beskriva sina tankar om arbetet i gränssnittet.

De beroende variabler som valts har inte varit tillräckligt känsliga. Frågan är om de verkligen ger ett bra mått på användbarheten hos ett program. Variabeln tid skapade en takeffekt då endast 8 av 18 försök klarades under tidsgränsen 30 minuter. Detta gör att variabeln tid inte längre kan avgöra användbarheten hos programmen. Inte heller variabeln antalet kompileringar ger en bra indikation på användbarheten i programmet, då försökspersonerna hade svårigheter i VAPS att kompilera om koden. Ett lågt mått på kompileringar gav då ett motsatt resultat.

5.3.3 Teoretiska synpunkter

Då expertis är domänspecifikt går det inte att dra några slutsatser om användbarhet för andra typer av experter förutom programmeringsexperter utifrån detta arbete. För att kunna avgöra huruvida resultatet kan generaliseras till andra programmeringssituationer behövs vidare tester. Det behövs mer kunskap om experterna och deras kognitiva scheman. Mer specifikt skulle vi behöva ytterligare kunskap om var gränsen går för deras scheman. Kan man använda samma scheman för programmering inom olika programmeringsspråk? Kan experternas scheman användas i andra programmeringssituationer än de är vana vid? Om detta är fallet så kan resultatet i detta arbete vara till hjälp vid utveckling av andra programmeringsverktyg. Om programmerarnas scheman däremot är väldigt specifika kan också resultatet av denna studie ifrågasättas då programmerarna i testet ställdes inför en ny situation med programmeringsverktyg de aldrig använt förut. Ytterligare behöver man studera vilken inverkan individuella egenskaper har för kognitiva scheman. Skapar alla programmerare liknande scheman i samma programmeringssituation? Om alla programmerare skapar olika scheman kan man inte generalisera designriktlinjer för att gälla alla expertprogrammerare. Om de däremot är likartade kan man skapa designriktlinjer gällande hur ett system avsett för expertprogrammerare bör vara utformat. Jag anser dock utifrån min undersökning att expertprogrammerare har likartade scheman för programmering. Under intervjuerna uttryckte de liknande åsikter om hur ett system bör vara utformat, samt hade liknande problem. Denna åsikt stöds också av de tidigare studier som gjorts på programmerare (McKiethen, et al., 1981; Soloway, et al., 1988). Resultaten från detta arbete skulle därför kunna ge intressanta frågeställningar till vidare studier om användbarhet och programmeringsexperter.

Detta arbete har även visat hur viktigt det ät att ta hänsyn till den förväntade användaren när man skapar ett datasystem. Alla programmen i testet var gjorda för att kunna programmera i, men försökspersonerna hade ändå stora problem med de uppgifter som fanns i testet. Detta kan bero på hur uppgifterna var utformade, men även på programmens utformning. Flera av de fel som uppkom var av sådant slag att man kan dra slutsatsen att det var programmen som

orsakade problemen. Ett tydligt exempel på detta är försökspersonernas svårigheter att kompilera sin kod i programmet VAPS. Det är inte rimligt att kräva att användaren ska förstå ett nytt system vid första användandet av detta. Men man bör sträva efter en så stor förståelse och så hög grad av användbarhet som möjligt. Om systemets uppbyggnad är naturlig för användarna utifrån vad de vill göra i systemet, samt enligt deras tidigare kunskap, krävs en kortare inlärningstid av systemet vilket i sin tur kan ge ekonomiska fördelar.

5.4 Förslag på fortsatta arbeten

Då studien innehåller korta och för en programmerare ganska onaturliga uppgifter vore det intressant att utföra en studie där de uppgifter som utfördes av programmerarna var sådana som var naturliga för dem. En mer omfattande studie av expertprogrammerarna skulle ge en djupare förståelse för deras arbetssätt, då möjligheten skulle finnas att studera deras hantering av programmeringsprocessen från början till slut. En sådan kunskap skulle ge större möjligheter att skapa användbara program som verkligen är anpassade efter experternas arbetssätt.

Då denna undersökning endast innefattar experter, och inte noviser eller mellananvändare, kan det vara svårt att avgöra om de användbarhetsproblem som framkommit under undersökningen är specifika för experter eller är generella. Om flera studier gjordes där även noviser och mellananvändare inkluderades skulle det vara möjligt att dra slutsatser om huruvida experter verkligen har specifika användbarhetskrav som inte är de samma som för andra användare. Om det visar sig att det finns skillnader mellan experter och noviser skulle det även vara intressant att ta reda på specifikt vad det är som skiljer experterna från de andra användarna och varför. Denna kunskap skulle kunna resultera i mer specifika designriktlinjer gällande programmeringsexperter.

Denna studie har inte kunnat avgöra huruvida experter har annorlunda användbarhetsproblem än noviser och mellananvändare. För att göra detta behöver experterna jämföras med noviser och andra användare. Studien har däremot funnit några specifika områden där jämförelser med noviser vore intressant. Experterna föredrog överblick över koden. Stämmer detta för noviser också? En jämförelse mellan experternas och novisernas hantering av fel vore också intressant, då detta kunde svara på om det är experternas kognitiva scheman som orsakat de problem som uppkom i felhanteringen under testet.

Referenser

Bevan, N (1997) Quality and usability: A new framework. I: van Veenendaal, E. & McMullan, J (red:er) *Acieving software product quality*, Nederländerna, Tutein Nolthenius.

Chase, W. G. & Simon, H. A. (1973) Perception in chess. *Cognitive Psychology, 4*, 55-81.

Chi, M. T. H., Feltovich, P. J. & Glaser R. (1981) Categorization and representation of physics problems by experts and novices. *Cognitive Science, 5*, 121-152.

Dillon, A. & Watson, C. (1996) User analysis in HCI –the historical lessons from individual differences research. *International Journal of Human –Computer Studies, 45*, 619-637.

Dumas, J. S., & Redish J. C. (1993) *A practical guide to usability testing*. Norwood, Ablex Publishing Corporation.

Gentner, D. R. (1988) Expertise in typewriting. I: M.T.H Chi, R Glaser & M.J Farr (red:er), *The nature of expertise* (1-21). Hillsdale: Lawrence Erlbaum.

Glaser, R & Chi, M.T.H (1988) Overview. I: M.T.H Chi, R Glaser & M.J Farr (red:er), *The nature of expertise* (xv-xxviii). Hillsdale: Lawrence Erlbaum.

Gulliksen, J. & Sandblad, B. (1996) Domain specific design of user interfaces. I: Gulliksen, J. (red) *Designing for usability –domain specific human computer interfaces in working life*, doktorsavhandling, Department of technology, Uppsala universitet.

Hoffman, R. R., Shadbolt, N. R., Burton, A. M. & Klein, G. (1995) Elicitating knowledge from experts: a methodological analysis. *Organizational behavior and human decision processes*, *62*(2), 129-158.

Kalyuga, S., Chandler, P. & Sweller, J. (1998) Levels of expertise in instructional design. *Human Factors*, *40*(1), 1-17.

Kujala, S. & Mäntylä, M. (2000) Studying users for developing usable and useful products. I: *Proceedings of the 1:st Nordic conference on computer-human interaction. (Stockholm, Sweden, 23-25 October)*, 1-11.

Lindgaard, G. (1994) *Usability testing and system evaluation.* London: Chapman & Hall.

Mayhew, D. J. (1992) *Principles and guidelines in user interface design*. Englewood cliffs: Prentice-Hall.

McKeithen, K. B., Reitman, J. S., Reuter, H. H. & Hirtle, S. C. (1981) Knowledge organisation and skill differences in computer programmers. *Cognitive Psychology, 13*, 307-325.

Patel, R. & Davidson, B. (1994) *Forskningsmetodikens grunder: Att planera, genomföra och rapportera en undersökning.* Lund, Studentlitteratur.

Proctor, R. W. & Dutta, A. (1995) *Skill acquisition and human performance*. Thousand Oaks, CA: Sage publications.

Prümper, J., Zapf, D., Brodbeck F.C. & Frese, M. (1992) Some surprising diffrences between novice and expert errors in compiterized office work. *Behaviour and Information Technology, 11*(6), 319-328.

Sasse, M-A. (1991) How to t(r)ap user's mental models. I : M.J. Tauber & D. Ackerman (red:er), *Mental models and human-computer interaction 2* (59-79). North Holland, Elsevier science publishers.

Shaughnessy, J. J. & Zechmeister E. B. (1990) *Research methods in psychology*. New York, The McGraw-Hill Book Co.

Soloway, E., Adelson, B. & Ehrlich K. (1988) Knowledge and processes in the comprehension of computer programs. I: M.T.H Chi, R Glaser & M.J Farr (red:er), *The nature of expertise* (129-152). Hillsdale NJ: Lawrence Erlbaum.

Streitz, N. A., Spijkers, W. A. C. & Van Duren, L. L. (1987) From novice to expert user: a transfer of learning experiment on different interaction modes. I: Bullinger, H. J. & Shackel, B. (red:er) *Human computer interaction- INTERACT '87*, North-Holland, Elsevier Science Publishers B. V.

Bilagor

Bilaga 1: Information till de medverkande

Tack för att du valt att deltaga i mitt försök!

Detta försök har till uppgift att undersöka hur väl anpassade programmeringsfunktionerna i de tre programmen VAPS, GL Studio och 3ds max är till en van programmerares arbetssätt. Försöket kommer att bestå av tre uppgifter, en i vart program.

Under försöket kommer en metod som kallas ”tänka-högt-metoden” användas. Det innebär att du, medan du arbetar i programmen, berättar för mig vad du gör och varför. Detta är för att jag ska kunna bedöma vad som händer i programmet även om jag saknar expertis i programmering.

Försöket kommer sedan att avslutas med en intervju, där jag kommer fråga om hur du tyckte det var att arbeta i de olika programmen.

Försöket går ut på att utvärdera programmen, din individuella prestation kommer ej att bedömas.

Försöket kommer att videofilmas för att underlätta för mig när jag ska analysera resultaten. Videofilmerna kommer ej att ses av någon annan person. Materialet kommer bara att användas av mig till denna undersökning, och inte i något annat syfte. Det slutgiltiga resultatet kommer inte att kunna kopplas till dig som person. Om du ändå känner någon form av olust eller obehag under försöket så har du när som helst rätt att avbryta.

Om det är något du undrar över kan du när som helst under försöket fråga mig. Tack än en gång för din medverkan!

/Anna Kardell

Bilaga 2: Intervjufrågor

- Först så skulle jag vilja veta din ålder, för statistiska skäl. (kön)

- Hur lång erfarenhet har du av programmering?

- Vilka program har du använt som programmeringsverktyg tidigare?

- Har du använt något av de tre programmen du har använt idag någon gång tidigare?

- Vad tyckte du om de tre programmen?
- Hur var de jämfört med varandra? (Var det något som var bättre än de andra?)

- Hur tyckte du det var att jobba i programmen?

    - VAPS
    - 3ds max Varför?
    - GL Studio

- Fanns det något du skulle vilja förbättra?

Bilaga 3: Transkriptioner av intervjuerna.

Teckenförklaring:

FP = försöksperson

I = intervjuaren

[kommentar] = förtydligande kommentar

Intervju 1

I: Då skulle jag först vilja veta din ålder.

FP: Jag är 36.

I: Bra. Hur lång erfarenhet har du av programmering?

FP: Eh.. det är, ja erfarenhet.. Arbetat med programmering har jag gjort i elva år.

I: Har du sysslat med programmering innan dess också?

FP: Ja, jo, utbildning hade jag ju, jag är utbildad i två år på högskolan i programmering, så det kan man ju också kalla en erfarenhet.

I: Ja. Vilka program har du använt som programmeringsverktyg tidigare?

FP: Det är väl mest visual studio, sen, jag har jobbat lite i simuleringssammanhang, fast det vet jag inte vad själva studion, eller själva miljön heter. Det var ju realtidsprogrammering.

I: Okej.

FP: Jag menar, det var ju inte såna här grafiska hjälpmedel som Visual Studio och såna här som vi har kört idag dårå.

I: Okej. Har du använt något av dom här tre programmen vi har använt idag någon gång förut?

FP: Nej, aldrig.

I: Nej.

FP: Eller Visual Studio har jag ju använt.

I: Ja.

FP: Det var vi ju inne i lite kanske.

I: Ja, okej. Vad tyckte du om de olika programmen?

I: Vi kan börja med GL-Studio som du använde först.

FP: Det var väl den som jag kände mig mest hemma i, men det kan ju ha att göra med att det var kopplat till Visual Studio, och att det var C++. Jag tror att.. eh.. om man börjat från början med dom andra så kanske man hade känt sig mera hemma. Det kändes nu som att man hoppar in nån stans mitt i och inte kunde men försökte göra nånting, om du förstår.

I: Ja.

I: Vad tyckte du om VAPS?

FP: Det var väl den av dom som var mest komplicerad, det var väldigt svårt att förstå vad man höll på med egentligen.

I: Ja, okej. Varför var det svårt att förstå?

FP: Ja, det var eh... mycket olika fönster och…

(Paus)

I: Det här sista programmet då, 3ds max?

FP: Ja, jag fick ju en koppling till vad allting… alltså dom är radiobuttons dom förstod jag ju vad de var för någonting, eh… jag tyckte inte att jag fick... det kanske var att jag inte riktigt gjorde rätt… att jag inte fick fram dom här lådorna med rätt färg. Hade jag fått… gjort rätt så att jag hade fått fram den röda lådan till exempel så hade jag nog förstått mer vad man kan göra med dom, att man kunde vrida på just den lådan och så. Nu satt jag hela tiden och körde huvet i väggen och försökte få fram den här med rätt färg.

I: Ja.

FP: Fast jag tror att det här är nog ganska lättarbetat det här progrmmet, om man lär sig då, vad man skulle kunna ha fram… eh… aaa…

I: Hur tyckte du programmen var jämfört med varandra?

FP: Ja hur menar du då? Hur dom var?

I: Ja, hur dom var att arbeta i.

FP: Arbeta i… em… VAPS var väl intressantast helt enkelt.

I: Varför det?

FP: För att eh… det fanns… Jag tror det fanns mycket möjligheter som inte jag kom in i, riktigt utan, men det fanns dom här olika att man kunde få fram dom här pilen och tryckknappen som var klara redan, att man… det verkar som man kan rita sina objekt och direkt provköra och se vad man hade gjort.

FP: Den här kändes mera som ja, den här vad heter den, den här listan…

I: 3d…

FP: 3d… att det var, eh…, att det var, att man inte har så mycket möjligheter att göra nånting i den här rutan och så då…

I: Finns det något du skulle vilja förbättra?

FP: Eh… i vahetere den här första...

I: GL Studio

FP: Där saknar jag helt klart kopplingen mellan hjälpen och programmet.

I: Mmm.

FP: Den här sista, 3d, där var det ju ganska så att när man hade sitt script att man kunde hoppa till hjälpen om det som man höll på med, som dom här radiobuttons och buttons, att man bara kunde trycka F1 och komma in på rätt ställe i hjälpen. Det är ju så jag är van att jobba, från Visual Studio.

FP: VAPS var väl inte alls så lätt att sätta sig in i som första-användare. Det var inte så självinstruerande, eller vad man ska säga…

I: Ja.

FP: Så jag får väl säga att den här GL, hette den GL Studio?

I: Ja, det första programmet.

FP: Det var väl det jag förstod mig på bäst. Men det kanske är beroende på att jag kände till Visual Studio.

I: Okej. Det var alla frågor jag hade. Då får jag tacka så mycket.

Intervju 2

I: Då skulle jag först vilja veta din ålder.

FP: Ålder… 23, just nu.

I: Hur lång erfarenhet har du av programmering?

FP: Sen 98… fyra år.

I: Fyra år, okej.

I: Vilka program har du använt som programmeringsverktyg tidigare?

FP: Eh… Borland Pascal C++. Och så C++ builder också.

I: Nåt annat?

FP: Edit plus.

I: Okej.

I: Har du använt något av dom här tre programmen vi har testat idag någon gång tidigare?

FP: Nej.

I: Inget av dom?

FP: Nej.

I: Vad tyckte du om de olika programmen?

FP: Eh… det första tyckte jag var förvirrande.

I: VAPS?

FP: Ja, så mycket småfönster. Det skulle vara lättare om dom satte ihop det till ett, bara ett program. Emm… Sen GL, det tyckte jag var helt okej. Fast det är lite bökigt att man måste ha VB för att kunna använda. Det är ju lite bökigt att man måste ha flera program, men det är helt okej. Och sen den här 3d Studio det är ju väldigt många funktioner att lära sig, om man nu ska kunna använda det riktigt.

I: Okej. Hur tyckte du det var att jobba i programmen, jämfört med ”vanlig programmering”?

ag är van vid att sitta och bara programmera, skriva kod så rent av, det känns mer som

FP: Jag tror det beror på vad man är van vid. J man vet vad som händer än när man sitter och, det är nog mycket lättare också när man har lärt sig ett sånt här verktyg helt så man vet vad man gör. Men när man bara sitter så här så är det mycket svårare och det tog mycket längre tid att göra dom små saker man ska göra, än att bara sitta med koden.

I: Okej.

I: Em… hur tyckte du de var jämfört med varandra, de olika programmen?

FP: Jag tyckte nog att GL var bäst, det var enklast, enklast att använda på något sätt. Det kändes mest lika C++ också, som är det jag har använt. Men jag tror 3d studio, jag tror det är mycket roligare om man lär sig, men VAPS tyckte jag inte om. Det tyckte jag var lite bökigt och svårt att förstå sig på vad man skulle göra i alla olika fönstren.

I: Okej. Finns det något som du skulle vilja förändra eller förbättra i de olika programmen?

FP: Ja, det är i så fall att sätta samman alla de olika fönstren i VAPS till ett, så man kan välja bland dom. Sen, jag vet inte riktigt, dom andra är väl, ja det är väl om man skulle kunna bygga i GL, bygga programmet utan att behöva VB.

I: Just programmeringen i de olika programmen, på vilket sätt skiljer det sig från hur du brukar arbeta?

FP: VAPS tyckte jag verkligen skilde sig för det var svårt att förstå vad dom menade med de olika variablerna och, ja det kändes så bökigt att man skulle ge anrop i det här verktyget till en funktion som man ändå skriver separat.

FP: Men sen ja, programmeringen i 3d den var ju helt okej. Nu var det ju script och det kändes lite mer, ja lite mer HTML-inriktat och det är ju inte jag så van vid, men det var ju enkelt, då skriver man bara koden och testar om den funkar, så det är nog bra.

FP: Och sen, programmeringen i GL, den var ju enkel, om man vet om man hållit på. Det var ju mushantering, dåråå. Då var det rätt enkelt att bara välja in det man vill ha.

I: Okej. Något annat du skulle vilja tillägga?

FP: Nej.

I: Då får jag tacka för det här.

Intervju 3

I: Då vill jag att du börjar med att berätta din ålder.

FP: 25

I: Hur lång erfarenhet har du av programmering?

FP: I arbetslivet två år, och tillsammans med högskola och gymnasium tror jag det är fyra år.

I: Tillsammans?

FP: Ja, tillsammans. Ja säj… nånting åt det hållet.

I: Vilka programmeringsverktyg har du använt tidigare?

FP: Det är Microsoft Visual Studio som vi såg alldeles nyss, och sen är det Borlands miljö från skolan.

I: Har du använt något av programmen vi har testat idag förut?

FP: Nej, det har jag inte gjort.

I: Okej.

I: Vad tyckte du om tre programmen?

FP: VAPS tyckte jag var helt, ja det var väl lite såhär… Det var väldigt svårt att komma in i det, hur man skulle göra i programmet. Det var inte helt intuitivt hur man skulle komma fram till saker och ting. Som en sån sak hur en if-sats skulle formuleras det var inte helt glasklart, det var det inte.

FP: Och 3d Studio max, det var väl bättre, jag såg ju bara ett skrap på ytan, vad jag kände dåråå, men det var inte helt svårt, de var bara att tuffa på och komme på hur man skulle göra.

FP: Och det här GL Studio tyckte jag var bäst, men där var hjälpen dålig. Den var verkligen i sämsta laget.

I: Okej. Tyckte du att du hade någon nytta av dina programmeringskunskaper i arbetet?

FP: Inte ett dyft egentligen. Men när man har programmerat ett tag så får man ju ett speciellt tankesätt när det gäller att behandla olika kod och sådär.

I: Mmm…

FP: Så jag får egentligen ta tillbaka det där med inte ett dyft, men om man säger att man har ju lite erfarenhet av det i alla fall. Man lär sig tänka på ”programmerare sätt”.

I: Hade man nån nytta av det när man arbetade i de här programmen?

FP: Ja, det var väl, mest i det här faktiskt, i GL Studio, tror jag.

I: Varför det, tror du?

FP: Ja, jag tror det kan vara kopplingen till Visual Studio, att man kan få fram allting i C-kod. Då ser man verkligen vad som händer och då kan man lättare lista ut hur man ska använda verktyget. Så det blir ju som att gå bakvägen för att lista ut hur man ska använd ett förenklat verktyg. Det kanske är fel sätt att använda det på.

I: Finns det nåt du skulle vilja förbättra i de tre olika programmen?

FP: VAPS ska bli mer användarvänligt, det är alldeles för många fönster att jobba i. Det var det ju ioförsig i GL Studio också, men det störde inte så mycket för det var lite enklare att använda. Men VAPS var rörigt och det var svårt att se vad man skulle ha alla grejor till. Det var mycket parametrar till mycket saker.

FP: Och 3ds max, det är svårt att kommentera det. Jag tryckte på några få knappar, det var inte så svårt att komma på vad man skulle göra.

I: Det var alla frågor jag hade.

Intervju 4

I: Jag skulle först vilja veta din ålder.

FP: Ålder, då måste jag räkna efter. Få se nu, jag är född 63, hur gammal blir man då? Jag är alltså… 38 va? Jag har inte fyllt år i år.

FP: Okej, 38, vi säger så.

I: Hur lång erfarenhet har du av programmering?

FP: Eh, Jag har jobbat här i… två år, vänta jag måste tänka. 3år, två och ett halvt.

I: Okej. Har du programmerat nåt innan dess?

FP: Nej Aldrig.

I: Aldrig?

FP: Nej.

I: vilka programmeringsverktyg har du använt innan?

FP: Som jag kan eller som jag har använt?

I: Som du har använt.

FP: J, vi använder ju ingen sån här miljö, utan vi kör ju en vanlig editor, och sen är det make-filen.

I: Okej.

FP: Edit plus heter editorn, men vi använder oss ingenting för att kompilera eller sådär, utan det är bara för att skriva kod. Sen på skolan så använde vi Borland. Borland, vad heter den?

I: Så då har du alltså erfarenhet från skolan också, programmering?

FP: Ja, det lilla vi gjorde där. Det var ju 120 poäng, men det var bara en liten del som var programmering.

I: Okej.

FP: Så det är ju inget. Det kan man ju inte säga.

I: Tillbaka till programmeringsverktygen.

FP: Ja, jag har använt lite hemma och fuskat lite med Borland C++.

I: Har du använt Visual Studio förut?

FP: Inte mer än den kursen, det var ju fem dar.

I: Har du använt något av de tre programmen som vi har testat idag?

FP: Nej, aldrig.

I: Vad tyckte du om dom tre programmen?

FP: Om man tar det här först…

I: GL Studio?

FP: Så tycker jag det verkar lite fånigt att hålla på och hoppa mellan två program.

I: Okej.

FP: Det finaste vore ju om man kunde kompilera direkt. Även om man använder Borland, eller Visual Studios kompilator, så skulle du kunna göra det ifrån GL Studio, helt klart. Även se hela koden, som genereras.

I: Dom andra programmen då?

FP: Ja…

I: VAPS?

FP: VAPS. Ja, jag vet inte, det är säkert bra när man lär sig, men helt intuitivt tyckte jag inte att det var.

I: Varför inte?

FP: Jag vet inte, eller det beror nog främst på att man inte känner igen begreppen.

I: Hur menar du då?

FP: Dom där, vad dom nu heter… Ja, vad hette dom nu?

I: Code1 och 2 kanske?

FP: ja, ioförsig det är ju nåt som dom har, det är ju default-värden, och ja, och det är ju alltid lättare om man får göra nåt från början istället för att bara hoppa in såhär, det är ju helt klart. Jag menar dom där listboxarna som fanns där på sidorna, där det stod ”event” ioförsig, det förstår man ju kanske vad det är, och ”object” och så vidare. Men det fanns ju andra där som hette ”frame och ”ATN” eller vad det nu hette, det har man ju ingen aning om vad det är.

I: 3ds max… script…

FP: Kommer faktiskt inte ihåg, jag ska titta på den igen…

[Försökspersonen sätter sig framför datorn och tittar]

I: Den uppgiften löste du ju ganska snabbt.

FP: Nu ska vi titta här.

FP: Eller ioförsig, det var nog den… förutom att man ska klicka på tusen ställen för at hitta olika grejor, men så är det ju i alla program. Men jag antar att man kan göra mycket mer än såhär.

I: Ja, absolut.

FP: Än bara att skriva in scripts sådär.

FP: Det är klart, kan man scriptspråket så, då är det väl inga större problem. Så då tror jag att det här var det mest lättförståeliga, det bästa av dom tre.

I: Varför tycker du så?

FP: Ja, men liksom, man fick upp koden på ett ställe och så bara skrev man i den. Det var inte så mycket hoppa mellan olika fönster kors och tvärs. Och framförallt inte mellanolika program.

I: Okej. Finns det nåt i de olika programmen som du skulle vilja förbättra, eller ändra på?

FP: Ja, den här 3… eller den här LG Studio, man ska inte behöva hoppa mellan programmen. Det verkar vansinnigt tycker jag. Ja, man kan ju lika gärna köra med make-filen.

I: Okej.

FP: Det är svårt att säga på en sån här kort stund, men generellt att, för alla tre, att det skulle vara mer att du får upp din ruta med kod, du editerar din kod, sparar och kompilerar bara med en knapptryckning, inte hålla på och hoppa runt. Här var man tvungen att köra ”run scrip” och då fick du upp en ruta där du var tvungen att välja script, och så vidare, va. Det är så många steg, bara för att göra det. A, man skulle kunna välja ett script och så när man kör run, så kommer alltid det scriptet.

I: Mmm

FP: När man sitter och.. när man skriver, ja man kompilerar ju ganska ofta, bara för att testa om det lilla man har skrivit funkar, så ska man göra det där var tredje sekund eller vad det nu blir, det blir rätt så mycket.

I: Ja.

FP: Men det gäller alla tre programmen tycker jag.

I: Okej.

I: Vad tyckte du att du hade svårigheter med? I GL Studio?

FP: Svårigheten var att, det som rörde till sig för mig, det var ju att jag tittade i Visual Studio så såg jag mer kod än vad jag såg här. Då vart jag osäker. Kan jag använda den här koden? Finns det någon vits med att jag inte ska se den? Antagligen så finns det väl det, men ändå så ville jag ju köra med den. Hade jag inte tittat där så kanske det hade gått mycket bättre.

FP: Det var liksom delar av koden som man inte visste vart de hörde.

I: Så du tyckte alltså det var svårt att få sammanhang?

FP: Ja…, ja.

I: Mmm.

FP: Och så tittar man i Visual Studio och så får man mer kod, ”jaha, det var ju inte så där”.

I: Om vi går över till VAPS då? Varför var det svårigheter där tror du?

FP: Ja, dels så var det ju det här att man skulle, dom här pilarna tillbaks. Så, om man gjorde något där så la han bara till, han tog inte bort det gamla. Ska det vara så egentligen?

I: Ja.

FP: Det tycker jag är jättekonstigt. Och det gick ju liksom inte att peta åt andra hållet, då vet jag inte vad som hände. Man fick inte över det som stod där från början i alla fall. Du klickade ju i de här cellerna, så stod samma sak där, och sen klickade man till nånting, du la ju till nånting där. Endera skulle det ha tagit bort det där eller så skulle det ha… eller det kanske var meningen… eller…jag vet ju inte. Det tyckte jag inte om. Sen när man skulle få opp det där då skulle man trycka på ett bock-tecken. Skulle det inte vara bättre om det var en pil där med, så det var lika. Sen var det ju ett kryss. Vad gjorde den då? Den tog tillbaks… neej… ja, det var ju inte helt logiskt det här, fast det lär man sig ju också, i och för sig, om man har gjort det några gånger. Så det kanske bara är en fråga om att lära sig. Det var ju inte direkt nåt man kom på utan att veta om det.

I: Du tyckte då att det inte var intuitivt, kan man säga det, att det stämmer?

FP: ja, det vart inte riktigt som man trodde att det skulle hända i alla fall, alltid.

I: Okej.

FP: Och sen verkar det lite småbuggigt.

I: Ja, det kan nog stämma.

FP: Jag menar, det lilla som jag gjorde nu, och ändå kom det konstiga saker. Vad händer då om man har en sån jättegrej?

I: 3ds max. Vilka svårigheter tycker du det fanns i det programmet?

FP: 3ds max, ja det var det med scripet?

I: Precis.

[paus]

FP: Svårt att säga men…

I: Ja, du kanske inte hade så många svårigheter?

FP: Allting är en fråga om att lära sig bra, vad kan man säga. Amen , du fick upp ett fönster, du kunde bara skriva ditt script och spara, även om det var många knapptryckningar var det väl ganska lätt…

I: Okej. Hur tyckte du det var att arbeta i programmen jämfört med ditt vanliga arbetssätt?

FP: Ja, man är ju van vid det gamla, det är klart, det är ju lättast. Men då har vi ingen riktig miljö som vi gör nu, utan skriver i en editor, och använder ett terminalfönster och kör make då. Och sen kanske om du ska köra programmet då, om det går att köra på din egen dator, så får du köra igång det. Det är klart det vore smidigare om du kunde göra det från samma ställe.

I: Om du tänker på, inte miljön du brukar arbeta i utan sättet du brukar arbeta på, om du brukar kompilera ofta eller andra saker. Gick det att jobba på det sättet i dom här programmen?

FP: Ja det går det, det tror jag. Har man lärt sig det så, förutom att det var väldigt många olika knapptryckningar, väldigt många steg tycker jag. Men så skulle ju den vara smidigare än vad jag gör idag. Det vore trevligare om man hade allt på ett ställe.

I: Vad tyckte du om programmen, jämfört med varandra?

I: Var det något som var bättre än dom andra?

FP: Ja, rent intuitivt så tycker jag att det där verkar bäst [3ds max] men samtidigt så är det ju mycket mer begränsat, vad jag har förstått. Du kan ju inte få ihop det med ”riktig kod”, eller vad man ska säga.

FP: Så att därför är väl det andra bättre, ur den synpunkten. Det där GL Studio bättre, trots allt.

I: Men du tyckte bättre om att jobba i 3ds eller?

FP: Ja, på grund av att du inte hoppade mellan applikationer.

I: Okej.

FP: Den där VAPS… Ja… Det har jag inte riktigt fått det för, komplettera in andra språk. Där gör du ditt vapsprogram. Kan du få ihop det, kan du använda det du gjort i andra applikationer.

I: Ja…, tror det…

FP: Ja, jag vet inte.

I: Okej.

FP: Var det inget mer?

I: Har du något mer att tillägga?

FP: men egentligen, för att få en riktig uppfattning om programmen så är en halvtimma för lite.

I: Mm.

FP: För det är ju inte förens men börjar förstå riktigt hur det funker som man kan säga att det är bra att programmera i.

FP: Men det kanske inte är det som är viktigt? Det kanske är mest mmi:t [(med mmi menas gränssnitt)] som är viktigt.

I: Ja..

I: Något övrigt?

FP: Nej.

I: Då får jag tacka för den här intervjun.

Intervju 5

I: Då skulle jag först vilja veta din ålder.

FP: 38

I: 38..

I: Hur lång erfarenhet har du av progammering?

FP: Då måste jag ju kunna räkna också. Få se… vad kan det vara? 16 år.

I: Okej, 16 år. Vilka program har du använt som programmeringsverktyg tidigare?

FP: Det beror på hur man delar in programmeringsverktyg, om du menar integrerad miljö eller om du menar kompilatorerna eller editorerna eller…

I: Tja.

FP: ja, rent generellt så här, ja, det är det vanliga. Nej, du måste nog precisera frågan mera.

I: Okej. Har du använt Visual Studio förut?

FP: Ja, det har jag gjort.

I: Det har du gjort. Vad brukar du använda för program när du arbetar?

FP: Det är väldigt olika, men senast så har det blivit mest Visual Studio, eftersom jag använder det i projektet.

I: Ja.

FP: Annars har jag ju använt det vanliga, skriva in text, compile, make.

I: Okej. Har du använt något av programmen du har testat idag nån gång tidigare?

FP: Nej, det har jag inte. Det som jag visste mest om var nog det här VAPS. Inte liksom hur man använder det, mera vad det är till för. Men ingenting sånthär som användargränssnitt, det är helt nytt alltså.

I: Vad tyckte du om dom olika programmen?

FP: Jag tror att det här 3ds max var lättast att komma in i. jag lyckades ju faktiskt lösa uppgiften, så att… eh… Så det kändes ju mer, man förstod intuitivt vilka grepp man skulle ta om jag säger så. Sen kommer jag inte ihåg hur det såg ut. Man tittade direkt i scriptet va?

I: Ja, precis.

FP: I jämförelse med den här, här har dom ju försökt att i viss mån gömma bort samanhanget för programmeraren innan, ja man kunde ju markera en hel rad där men…ja det känns som dom har misslyckats med att gömma rätt saker.

I: Ja.

FP: jamen den kändes mest intuitiv faktiskt: Det påminde mycket om Visual Basic till arbetssätt… faktiskt…

FP: Den här VAPS bryter mest, eller avviker mest från dom övriga. För GL Studio har i alla fall nån form utav, någon slags objekt-tänkande också. Även om jag inte förstod riktigt hur det hängde ihop där riktigt.

I: Ja, okej. Vilka svårigheter tycker du det fanns i programmen? Vi kan börja med VAPS då.

FP: Ja, rent generellt, så, vilka arbetsmoment du ska göra, hur det hänger ihop, sammanhanget. Ja menar, varje grej kan man ju liksom förstå att, ja det här är min tabell över en statement, det har man ju liksom läst om och använt i olika projekt och så vidare, men hur hänger det ihop med det övriga? Det ser man inte. Det är precis som om dom har lagt ett skynke över vissa bitar, så man liksom inte kan titta efter hur det funkar och så där.

I: Okej. Varför är det en svårighet tror du?

FP: Det är ju… om man nu har programmeringsbakgrund, kunskap om hur det funkar va, genom att lägga på ett dåligt mmi så kan du ta bort information som är viktig, så att

säga, eller svårnåbar. Man måste ju göra mmi:t efter det problem man ska försöka lösa.

FP: Personligen så tycker ju jag, isåfall som ett ritprogram så här, där det skulle vara direkt i det här fönstret då, om jag trycker på en knapp liksom, då får jag liksom upp, jaha vad händer om jag då trycker på den här. På den nivån. Nu är det massa liksom, spridda skurar, lite grann va. Så man måste ha förklarat, innan här va, hur dom olika fönsterna hänger ihop. Annars har man ju ingen chans att förstå vart man ska in och dutta nånstans. Finns ju inte en möjlighet, förutom att prova sig fram då.

I: Okej, om vi tas 3ds max då? Du kanske inte hade så mycket svårigheter.

FP: Nej då. Det lilla man skulle göra, det gick liksom att gissa sig fram till hur man skulle göra.

FP: Det är för svårt att kräva det här VAPS, på vad var det, en halvtimma.

I: Ja.

FP: Man måste ju ha mer på sig bara att läsa, jag menar man måste ju läsa ett tag för att sätta sig in i texten, hur hjälpdokumentet var uppbyggt och så vidare. Då har man ju liksom bränt en halvtimma. Enda sättet att ta sig fram är liksom att försöka gissa sig fram med den erfarenheten man har med sig.

I: Ja, precis.

FP: Jag tror nog att det där 3ds max i jämfört med GL Studio, GL Studio är liksom mera generellt. Där kan man programmera upp och göra liksom vad man vill dåråå. Dock på bekostnad av en högre inlärningströskel. VAPS kräver nog en viss vana att jobba i om man ska få ut det mesta utav det. Sen vet ja inte om den här modellen man har valt för att den heller med tillstånd och tillståndsförändring, om det är korrekt sätt för det här problemet, det vet jag inte. Jag har inte hållit på med så mycket ritverktyg som sådant, sen tidigare. Så det kan jag inte uttala mig om.

I: Okej, GL Studio då?

FP: Tja…

I: Varför var det svårigheter att jobb i det programmet?

FP: Ja, det var likadant som det här också, det är inte uppenbart vad man ska göra. Där så tycker jag att det skulle vara så att man hela tiden jobbar, om man nu har ett hjälpmedel för att rita saker, då ska arbetet var centrerat eller koncentrerat kring det man ser, va, typ.

I: hur var det att jobba i programmen jämfört med ditt vanliga arbetssätt?

FP: Tja, jag håller inte på med ritprogram så jag har ingen uppfattning. Eller programmerbara ritverktyg. Ritverktyg han man ju använt, till dokumentation och sådant. Men inte kopplat till beteende bakom, dåråå. Har jag ingen som helst erfarenhet av.

I: Det arbetssätt du har när du programmerar, går det att använda dem i de här programmen?

FP: ja, det tror jag absolut.

I: Okej.

FP: Ja, det är ju avancerade verktyg för att beskriva beteende på grafiska objekt. Det stöd man får är just att man slipper i koden direkt med kommandon rita upp det, man ser ju hur det ser ut, det är ju den hjälp du får. Det man missat är kopplingen mellan liksom det här [pekar på objekten på datorn] och programmeringssidan.

I: Det var alla frågor jag hade. Har du något du vill tillägga?

FP: Nej.

I: Då tackar jag för denna intervju.

Intervju 6

I: Då vill jag först veta hur gammal du är.

FP: 32.

I: Hur lång erfarenhet har du av programmering?

FP: Ska man räknade att man började i skolan eller, började här?

I: Ja, erfarenhet, sammanlagd.

FP: Nio år då.

I: Vilka program har du använt som programmeringsverktyg tidigare?

FP: Tja, programmeringsverktyg, det beror liksom på hur du definierar det. Som Visual Studio, det är ett programmeringsverktyg, men codewrite anser jag inte är något programmeringsverktyg, för det är ju ingen… eller ioförsig, Visual Studio är inte mera programmeringsverktyg än codewrite egentligen, du har ju lite hjälp med variabelnamn och sånt men, inte på det här sättet att du ritar objekt och skriver kod till dem. Men annars är det Visual Studio. Och Unix.

I: Har du använt något av programmen vi testat idag någon gång tidigare?

FP: Nej, inte nån av dom här, Visual Studio har jag använt, men inte de andra.

I: Vad tyckte du om programmen?

FP: Ja… tyckte och tyckte…

FP: Ja, ska man börja med ett nytt system så vill man ju ha ett bra hjälpsystem och då är ju HTML-varianten att föredra. Pdf är ju rena rama katastrofen. Det var det ju nån som hade, vilket program det nu var.

I: Det var GL Studio.

FP: Då var det ju 3ds max som hade den där, och det gick ju ganska snabbt. Så den var ju bra då.

I: Om var programmen ett och ett då, vad tyckte du om GL Studio.

FP: Jadu, vad ska jag säga om det. Nu var det inte självklart hur dom här kopplingarna skedde, man gör ju nånting, kopplar en knapp till en händelse. Då hade jag lite problem att fatta, ha, ska koden in här eller?

FP: Normalt sett så kanske man börjar på en sån här tutorial eller nåt . Det är rätt avgörande hur den den ser ut, för kommer man helt ny sätter man sig ju inte och tok-kodar.

FP: Annars så. Fönster hade du ett gäng att titta i. Det är lite olika som man tycker, tycker man om att ha olika fönster så om man vill växla, om man har ett fönster och växlar mellan olika vyer i det fönstret.

FP: Det var ju rätt så hyfsat i alla fal, det fanns ju i en lista vad du kunde göra, så slapp du sitta och klura.

I: 3ds max.

FP: Ja den är ju inte så… krånglig egentligen.

Det var ju liksom kod liksom rakt av och den var ju inte så svår att lösa.- och sen bra hjälpfil så att man hitta bra. Man fick ju skicka filen för att provköra den. Om man ska göra det ett par gånger så blir det lite jobbigt. Det skulle finnas en knapp för den, å man kunde bara trycka på knappen och provköra igen.

FP: Annars så, det lilla jag såg, av det så var ju rätt okej. Det var ju rätt så enkel miljö det var ju bara texteditor rakt av, men det är ju tänkt så i små kort scripts funktioner så det är väl helt okej.

I: Okej: VAPS.

FP: Fortfarande ingen höjdare med pdf-fil i dokumentationen. Många fönster.. Sen…

[försökspersonen tittar på datorn]

FP: Aaa, mycket fönster, nästan väl mycke fönster asså. Ja, hm, jag tycker det ser lite rörigt ut.

I: Hur var det att arbeta i dessa program jämfört med ditt vanliga arbetssätt?

FP: Om du gör nånting grafiskt så, nu jobbar jag mest i QT. Det finns i Visual Studio, har du en liten toolbar som du klickar i , då får du QT-designern. Det är ett mdi-dokumet. Arbetsyta, toolbar har alla möjliga knappar du kan lägga in. I vänsterkanten har du liksom, någon parameter du kan sätta på nått objekt. Så då går vi in där och skapar ett projekt och lägger till lite… eh… formulär och en dialogruta och bitmappar och edit-fält, sätter an på dem, sen kan du, du kan ju skriva kod där också, men det är inte så bra. Skriv ingen kod i verktyget, för då genererade det ned i… i… Aaa, vi kan säga att du gör ditt interface, sparar det, går över i Visual Studio, då har du.., då skriver verktyget nåt som kallas en UI-file, user interfce-file. Den kör du genom en, i Visual Studio, en make och en user interface compiler, då kommer den automatiskt att gör om den här till C-kod.

I: Hur var det att programmera i de här programmen?

FP: Nu var det lite svårt, eftersom då ska man ju egentligen börja från början. Det är liksom det knöliga ibland, att sätta upp själva projektet eller vad man ska säja. Miljön runt den och få den och fungera bra. Det kan vara knöligt. Sen lägga till knapparna tyckte jag inte var så svårt. Det är ju lite så det handlar om, det kan vara en halv dags jobb bara att få så man kan kompilera.

I: Ja, precis. Men det är ju inget man kan ha med i ett sånt här test.

FP: Nej precis, det blir ju lite för komplicerat. Annars var det inge svårt. Det var kanske den där första. [GL Studio] Där kopplade inte riktigt jag, hade inte riktigt begreppsvärde, vad det här input-device, liksom hur kopplas den till de här objekten. Det var ju den svåra biten där, och sen att skriva koden det är ju inga problem.

I: Det var alla frågor jag hade. Då får jag tacka för mig.