SCHOOL OF ECONOMICS
AND MANAGEMENT
Lund University

Department of Informatics

Fem hjul och en propeller

-en explorativ studie av alternativ till skrivbordsmetaforen

Lunds Universitet
Informatik

Kandidatuppsats, 10 poäng, inom det systemvetenskapliga programmet.

Framlagd: Juni-2007

Författare: Jakob Nilsson
Joel Sannerstedt

Handledare: Lars Fernebro

Examinator: Erik Wallin
Mia Sassen

LUNDS UNIVERSITET
Institutionen för informatik 070613

Fem hjul och en propeller

-en explorativ studie av alternativ till skrivbordsmetaforen

©Joel Sannerstedt
Jakob Nilsson

Kandidatuppsats – VT 2007
Omfång: 44 sidor
Handledare: Lars Fernebro

Abstract

I dagens samhälle ökar datoranvändandet stadigt och datorerna blir snabbare och snabbare. Trots detta står utvecklingen på ett område i det närmaste still, gränssnitten till våra operativsystemet. Den stora majoriteten av dagens gränssnitt bygger på samma principer som när de introducerades för över 20 år sedan, dessa principer kallas för WIMP vilket står för window, icon, menu och pointing device. Då operativsystemet kan sägas vara basen för allt arbete med en dator och till väldigt stor del styr effektiviteten i användningen, kan man undra varför inte mer har hänt på detta område. Syftet med vår undersökning är att visa på att det dels finns tydliga användbarhetsproblem med dagens dominerande gränssnitt, samt att det finns flera alternativ som erbjuder annorlunda lösningar. Vi utför därför en explorativ studie över ett antal alternativa gränssnitt som ännu inte fått genomslagskraft, och redogör ingående för dessa. Vi genomför även en värdering av både WIMP och alternativen ur ett användbarhetsperspektiv. Denna värdering baseras på en metod som kallas heuristisk utvärdering, vilken senare modifieras för att bättre passa vårt syfte. Efter vår analys kommer vi fram till att det finns tydliga problematiska områden kring WIMP-baserade gränssnitt. Vi redogör vidare för att alternativen presenterar lösningar som bättre stämmer överens med teori kring användbarhet.

Nyckelord: WIMP, användbarhet, heuristisk utvärdering, Archy, OLPC, Croquet, Type Managers

Innehållsförteckning

Illustrationsförteckning

Tabellförteckning

1 Introduktion

1.1 Inledning

Idag, när man slår på en vanlig hemdator möts man i regel av en färgglad representation av ett skrivbord. På skrivbordet möts vi av bilder av mappar och filer med vilka vi sedan starta applikationer eller manipulera data på olika sätt. Så har dock inte alltid varit fallet. Tidigare system var kommandobaserade, datorerna styrdes då genom att användare matade in olika parametrar (van Dam 1997). På 1970-talet kom den första datorn med vad vi idag refererar till som ett modernt gränssnitt, utvecklat av Xerox Palo Alto Research Center (PARC) (Cooper 1995, s. 55). Denna typ av system kallas idag WIMP [Window, Icon, Menu, Pointing device] efter de delar som systemet bestod av (Green & Jacob 1991, refererad i Morrison 2000). Det var inte långt senare som datorn blev verkligt tillgänglig för den breda allmänheten, nämligen 1984, då Apple lanserade sin Macintosh och med detta, ett system baserat kring metaforen av ett skrivbord. Samma metafor som än idag är helt dominerande på marknaden.

I en marknad där all utveckling sker försvinnande snabbt, kan det tyckas konstigt att själva grunden, styrsystemet, inte har förändrats påtagligt under en så lång period. Moore's law konstaterar att datorer blir dubbelt så snabba var 18e månad. Man kan då undra hur det kommer sig att datorer som är så många gånger snabbare än de var 1984, inte har ett annorlunda gränssnitt. Är det möjligen så att WIMP är en ultimat lösning, som är tillräckligt flexibel för att inte behöver bytas ut? Utvecklare bakom ett projekt, kallat Croquet, för fram en annan tänkbar förklaring och slår samtidigt fast stagnationen inom tänkandet kring gränssnitt

> The emergence of software monopolies removed any encouragement to innovate on the platform. Compare a modern PC of 2004 to the first Macintosh shipping in 1984 and the major difference you will see is color. (Croquet Consortium, 2007b)

Utvecklandet och nytänkandet kring gränssnitt för operativsystem har alltså varit väldigt begränsat p g a det monopol som uppstått på markanden. Det är i så fall möjligt att WIMP-baserade system med skrivbordsmetaforen har allvarliga brister, men att användare inte har någon möjlighet att välja något annat styrsätt då det under lång tid inte skett någon utveckling på området.

1.2 Problemområde

Det är egentligen inte så anmärkningsvärt att WIMP har en sådan dominans på operativsytemsmarknaden. Idén tvecklads och fick ett väldigt genomslag, efterföljare som vill ha en del av marknaden försöker då göra samma sak fast med modifikationer. Efter en period säljs majoriteten av datorer med denna typ av system. Dess egenskaper associeras snart med datorn. På samma sätt som det idag skulle vara svårt att sälja en bil med propeller och fem hjul oavsett hur mycket bättre den var, blir det svårt att sälja en dator som inte startar upp med ett skrivbord som standard. Detta tillstånd brukar inom ekonomiska ämnen beskrivas som att det finns en dominant design. En produkt har en så dominerande ställning att efterföljare måste göra väldigt liknande

produkter för att kunna komma in på marknaden.

Marknaden för gränssnitt till operativsystem har alltså varit inne i en period av väldigt lite nytänkande och utveckling. På senare tid har dock saker börjat hända. Nya system som har gränssnitt vilka bygger på fundamentalt annorlunda principer från WIMP-baserade system har börjat dyka upp. Utvecklarna till dessa menar ofta att det finns allvarliga problem med dagens dominerande gränssnitt, något som de vill komma tillrätta med genom sina egna system. Kritiken kan delas in i två huvudgrupper. Den första gruppen är den som består av kritik mot ett specifikt WIMP-baserat system, t ex Windows XP, och innebär att systemet lider av ett antal problem av olika typer. Den andra gruppen innehåller kritik kring att WIMP som princip är förlegad och därmed har gjort sitt på operativsystemsmarknaden. Med detta menas att själva grundidén lider av stora brister som gör arbetet med en dator ineffektivt och otillfredsställande. Kritiken saknar oftast teoretisk grund, och baseras istället på subjektiva åsikter eller personliga erfarenheter. Vi tycker oss dock finna starka kopplingar till begreppet användbarhet. Att designa system med hög grad av användbarhet är idag självklart för de flesta systemutvecklare. Begreppet har tydliga definitioner och är välkänt idag. Så var dock inte fallet när WIMP-system och skrivbordsmetaforen först skapades. Det ter sig naturligt att när nya system idag skapas, så görs det utifrån andra teoretiska ramar och idéer.

1.3 Syfte

Huvudsyftet med vår studie är att visa att det finns andra sätt att använda våra datorer på än dagens WIMP-baserade system med en metafor av ett skrivbord med tillhörande fönster. Detta är ett område som det finns lite skrivet om sedan tidigare och något som vi tror att många inte ens reflekterar över. Vi vill även visa på att det finns ett behov av och utrymme för nytänkande kring utvecklande av gränssnitt för operativsystem genom att studera de principer WIMP-gränssnitt baserar sig på ur ett användbarhetsperspektiv. Vidare vill vi på detta vis skapa förutsättningar för vidare forskning kring området. Utifrån detta kan våra konkreta frågeställningar formuleras som.

> *Hur ter sig dagens WIMP-baserade gränssnitt ur användbarhetssynpunkt?*
>
> *Finns det idag alternativa gränssnitt, och hur skiljer de sig från de WIMP-baserade med hänsyn till användbarhet?*

1.4 Begreppsdefinitioner

Vi kommer nedan att gå igenom några begrepp som är väldigt centrala för vår uppsats.

1.4.1 Operativsystem

Operativsystemet tillhandahåller funktioner som låter användaren styra, men även interagera, med datorn, med hjälp av ett gränssnitt. Man kan se operativsystem som ett grundläggande program för att starta upp och hantera program men operativsystemen hanterar även filer. Följande definition av operativsystem, hämtat från dictionary.com (2007) har vi funnit användbar i vår analys: ”The collection of software that directs a computers operations, controlling and scheduling the execuion of other programs, and managing storage, input/output and communication resources.”

1.4.2 Gränssnitt

Alla som har jobbat vid en dator har någon gång interagerat med ett gränssnitt. Det finns olika typer av gränssnitt, men det är grafiska användargränssnitt (Graphical User Interface [GUI]) som vi håller fokus på. Ofta förekommer det att vi använder begreppet *gränssnitt* för att beskriva grafiska användargränssnitt. Definitionen har vi hämtat från searchvb.techtarget.com (2007) , där det beskrivs som ett skal vilket ligger ovan ett operativsystem för att användaren lättare ska kunna interagera med datorn. Ett typiskt system som inte använder något grafiskt gränssnitt är DOS operativsystem, som fortfarande går att nås från Windows. Ett system som använder ett grafiskt användargränssnitt använder sig ofta utav metaforer, och skrivbordsmetaforen är mycket vanligt förekommande i våra operativsystem. Ett mellansteg mellan kommandotolken, så som DOS, och grafiska gränssnit är de s k menybaserade gränssnitten som tillåter användaren att interagera med menyer, med hjälp av ett pekdon. Men det är de grafiska gränssnitten som vi kommer att fokusera på i denna uppsats.

1.4.3 WIMP

Den vanligaste typen av gränssnitt för operativsystem idag är de som kallas WIMP.

Windows. Programmen som vi kör i våra WIMP-baserade operativsystem, innehåller två olika sorters fönster. Dels huvudfönster, som kan innehålla dokument och dels underordnade fönster, så som dialogrutor. Moderna fönsterhanterare bygger en överlappningsfunktion, där vi kan flytta ett fönster ovan ett annat. Fönster kan se olika ut, men har alla den egenskapen att det ska hjälpa oss att fokusera på funktioner och orientera oss i olika program. Cooper (1995) beskriver fönsters syfte som rum i ett hus där varje rum i huset fyller ett eget syfte. Huset i det här fallet representerar själva programmets huvudfönster och varje rum är programmets övriga fönster, så som dokumentfönster och dialogrutor. Cooper understyrker vikten av att varje fönster ska ha ett unikt syfte och gör jämförelsen med rum och fönster. I verkliga livet bygger vi inte ut vårt hus med fler rum, om rummet i sig inte fyller något syfte. Inte heller bör vi lägga till fler fönster i vårt program, om det inte fyller något syfte.

Icons. En ikon i datorsammanhang är en symbol för exempelvis en fil, funktion eller ett program. Ikoner förekommer ofta i ett programs verktygsfält, och då vanligen i form utav en tredimensionell bild, vilken man kan klicka på. Visuella ikoner kallar Cooper för "buttcons" då de fungerar som en kombination av en knapp och ikon.

Syftet med ikoner är att användaren enkelt ska känna igen sig i programmet och dess funktioner. Exempel på en vanlig ikon, är den som symboliserar att man ska spara data på hårddisken. Även om symbolen, en traditionell 3.5" diskett idag är föråldrad och knappt används längre, lever den kvar som symbol för att spara data.

Menu. Menyer låter användaren komma åt fler funktioner än de som är direkt synliga i programmet. Med andra ord är de direkt motsatsen till verktygsfält, som är direkt kan nås, Löwgren (1993).

Pointing device. P:et har även definierats som: pointer, pulldown menu m fl. Vi har valt att översätta pointing device till pekdon, vilket är vanligt förekommande i svensk litteratur. Det klassiska och vanligt förekommande pekdonet kallar vi för mus. Musen är datorkomponenten som låter användaren styra vad som händer på skärmen. Genom att flytta musen, flyttar man också pekaren på skärmen. De grafiska gränssnitten inom WIMP tillåter användaren att exempelvis flytta fönster eller dra filer, s k *drag-and-drop*. Detta både är och har varit, en ytterst viktig egenskap för att jobba med applikationer. Även om en del applikationer inte kräver att man använder ett pekdon, är det desto mer nödvändigt när man jobbar i operativsystemet.

Det finns alternativ till musen. Löwgren (1993) har identifierat några: *touchscreen*, *trackball*, *eye tracking*, *speech input*, *character recognition*, *lightpen*, *joystick* m fl. Många gånger passar olika anordningar till olika arbetsuppgifter. Exempelvis är det många som föredrar ritplattor framför en mus, när de jobbar med viss typ av grafik på datorn.

1.4.4 Skrivbordsmetaforen

Ett begrepp som inte ingår i akronymen WIMP, men som är tätt förknippat med paradigmet och på många sätt är central i dagens operativsystem är skrivbordsmetaforen. Skrivbordsmetaforen innefattar ikoner, filer, mappar, nätverk och så vidare. De flesta operativsystem som bygger på WIMP, har skrivbordsmetaforen som grund. Många anser att Macintosh är grundare till metaforen, detta p g a att det var Apples introduktion av Macintosh som gjorde den känd. Cooper (1995, s. 63) anser att skrivbordsmetaforen kan ses som en global metafor. Vidare menar han att den kan i sig, ses som ett ramverk för alla underliggande metaforer, så som mappar, och filer. Med global metafor menar Cooper en metafor som spänner över hela systemet och som agerar grundläggande för de övriga metaforerna.

1.5 Avgränsningar

Från början var vår avsikt att försöka finna alternativ som skiljde sig helt från WIMP. Dock fann vi snart att det var i princip omöjligt. Nästan alla system som är under utveckling idag använder sig av någon form av *pointing device* för att låta användaren utföra olika funktioner. Vi har därför valt att inte fokusera på just denna del av WIMP, även om vi i fortsättningen kommer använda det som begrepp. Det är inte heller själva operativsystemet, hur det hanterar hårdvara och bakomliggande processer vi är ute efter att undersöka. Det är istället den del som möter användaren när denne startar sin dator, alltså själva gränssnittet. Vidare, när vi talar om gränssnittet, så är det främst den metafor av ett skrivbord som är det centrala. Man kan tänka sig system som har fönster, menyer, ikoner som inte är byggda kring en bild av just ett skrivbord, även om det i princip inte förekommer.

I urvalet av alternativa gränssnitt, kommer vi endast att diskutera lösningar som har en dokumenterad historia och där användbarhet är i fokus. Eftersom de skiljer sig helt från WIMP, kan det verka som radikala förändringar vid första anblicken, men vi vill påpeka att det är principerna bakom som är intressanta för oss. Alternativen har valts utifrån en bedömning av relevans samt hur väl fungerande prototyp som funnits tillgänglig.

1.6 Målgrupp

Studien riktar sig i princip till alla som använder en dator. Men framförallt till personer som har ett intresse när det gäller nytänkande inom gränssnittsutveckling och är nyfiken på framtida lösningar för att hantera information på en dator. Vi tror även att uppsatsen kan fungera som underlag för vidare diskussion kring ämnet och förhoppningsvis vara en ansats till att skapa nya intressanta diskussioner om gränssnitt i framtidens operativsystem.

1.7 Disposition

Vår studie skiljer sig från en vanlig uppsats i den meningen att vi inte har genomför en traditionell empirisk undersökning eller litteraturstudie, och därmed skiljer den sig en del i sin disposition. I kapitel två beskriver vi vårt angreppssätt i en metoddel. Vi har funnit det av extra stor betydelse att beskriva hur vi går tillväga för att uppnå vårt mål, eftersom vår studie inte innefattar den traditionella empirin. Kapitel 3 innefattar teoridelen, där vi presenterar en modell med ett antal punkter kallade *heuristics* och som sedan ligger till grund för vår bedömning av användbarheten såväl i WIMP-baserade system som i de alternativa system vi väljer att ta upp. I nästföljande kapitel, det fjärde, presenterar vi designprinciperna för WIMPs uppbyggnad. Dessa principer gäller även för utveckling av mjukvara. I det femte kapitlet inleds vår analys och vi kan utifrån vår modell analysera principerna i WIMP. Kapitel sex behandlar sedan de gränssnitt som vi funnit i vår explorativa studie. Dessa redogörs för grundligt och diskuteras sedan ur användbarhetssynpunkt. Det sjunde kapitlet utgörs av en sammanfattande diskussion.

2 Metod

Huvudsyftet med uppsatsen var att undersöka vilka alternativ till WIMP-baserade gränssnitt som finns, samt relatera dessa till teori kring användbarhet. Centralt var då att först slå fast vad som menas med detta begrepp samt, som kontrast även studera detta ur ett användbarhetsperspektiv. I objektorienterade termer så var det den abstrakta klassen WIMP vi ville ringa in, inte dess konkreta generaliseringar. Att beskriva något så stort och omfattande som ett operativsystem på ett koncist sätt innebär nog med problematik. Att ringa in och kritisera generella egenskaper hos flera system har heller inte varit möjligt för oss i denna studie. Förutom att det finns flera olika typer av system som bygger på denna princip, som dock ser olika ut, innehåller varje system ett otal funktioner som är problematiska att gruppera på ett begripligt vis. Att arbeta med och observera ett av dessa system för att sedan försöka sammanfatta dess funktion och innebörd har vi inte funnit genomförbart inom våra tidsramar. Dessutom skulle vi varit tvungna att studera flera olika system och leta efter gemensamma nämnare för att hitta generella egenskaper. En egen observation hade alltså inte varit en särskilt realistisk väg att nå fram till innebörden av WIMP.

Vi har istället valt att, med utgångspunkt i WIMPs principer, studera de designprinciper som erbjuds från några av de största utvecklarna av operativsystem. Dessa presenteras i olika form, men är skrivna som en slags grund utefter vilken systemen ska utvecklas. Vi märkte snabbt hur lika dessa var, systemen bygger på så gott som exakt samma principer, men kan fungera kompletterande på vissa områden. Dessa dokument erbjuder ett enkelt sätt att få fram tillverkarnas bakomliggande tankar kring hur systemen ska fungera. Även om de faktiska systemen skulle avvika något från dessa principer så är det de generella tankegångarna som är intressanta för vår uppsats, snarare än faktiska buggar och dylikt. Vi menar att de därför är ytterst lämpade att fungera som vår definition av innebörden i begreppet WIMP. När denna definition var färdigställd visste vi vad vi skulle utgå ifrån, d v s vad det var vi skulle finna alternativ till.

För att ta reda huruvida det finns några alternativa gränssnitt, och vilka dessa kan vara, utförde vi något som i litteraturen benämns som en explorativ studie (jmf Grønmo 2004, Halvoirsen 1989). Med detta begrepp avses undersökningar som genomförs utan att avsikten är att dra slutsatser om någon större generell företeelse. Urvalet av enheter är inte sammansatt på ett sätt som ger underlag för en systematisk generalisering. Istället väljs undersökningsenheterna ut på ett godtyckligt eller pragmatiskt sätt. Grønmo (2004) skriver att avsikten med denna typ av undersökningar kan vara att genomföra provisoriska studier kring ett område som inte är utförligt utforskat sedan tidigare. Målet är då att skapa insikter kring ett område och ge underlag för vidare studier. Denna typ av studier, även kallade pilotundersökningar, har ofta kvalitativa ansatser och baseras ofta på ett relativt litet urval (Grønmo 2004, s. 91).

Vår explorativa studie bestod främst i sökande på internet, efter olika system som har ett gränssnitt med en utgångspunkt som skiljer sig från den WIMP-baserade. Många system föll bort då de vid en närmare undersökning visade sig vara alltför lika, även om de vid första anblick tycktes erbjuda nya intressanta lösningar. Vidare ansåg vi att många idéer, om än väldigt spännande, låg alltför långt bort från att realiseras för att kunna undersökas. Kvar hade vi då ett fåtal system, som var väl dokumenterade, hade flera intressanta aspekter, samt någon form av prototyp.

När vi samlat dessa system, samt slagit fast innebörden i WIMP ville vi utvärdera dessa med hänsyn till användbarhet. I litteraturen skiljer man normalt mellan två typer av metoder för detta, inspekterande och testande, eller analytiska och empiriska om man så vill (mer om detta i 3.1.5).

Liksom fallet med definitionen av WIMP, fann vi det inte heller här lämpligt med empiriska tester eller observationer. Att placera en användare framför en windowsdator och be dem utföra ett antal uppgifter är fyllt med problematiska invändningar. Som vi tidigare nämnt är datoranvändandet idag så gott som synonymt med WIMP-baserade gränssnitt. Skrivbordsmetaforen är något som gemene man är väldigt bekant med utan att man för den skull reflekterar särskilt mycket över densamma. I Sverige är det 78% av befolkningen som har tillgång till en stationär dator och 65% som använder datorer dagligen (SCB 2006). Denna typ av tester skulle i vårt fall löpa en stor risk att bli missvisande.

Istället beslutade vi oss för att skapa ett teoretiskt ramverk utifrån vilket låg till grund för vår analys, både av WIMP samt av de alternativa gränssnitten. Modellen baserade vi i huvudsak på en metod som kallas expertutvärdering, eller heuristisk utvärdering (Nielsen (1993b). Metoden bygger på att en eller flera personer med god kännedom om användbarhet utvärderar ett system utifrån ett antal på förhand givna principer, *heuristics*, och på så sätt identifierar tänkbara problem. Denna modifierades sedan och kompletterades med övrig litteratur för att passa vårt syfte. Vi redogör närmre för denna process under vårt teoriavsnitt då den heuristiska utvärderingen är förankrad i litteraturen kring användbarhet.

Utifrån vår teoretiska modell genomförde vi sedan en kvalitativ bedömning av de principer som den WIMP-baserade gränssnitt är utvecklade efter. Målet var inte att genomföra en oomtvistad analys utan snarare att ge en bakgrund och påvisa att det finns ett behov av nytänkande och att de idag ledande systemen har brister i sitt tänkande kring användbarhet.

3 Teori

3.1 Metaforer och konceptuell modell

Ett dominerande tema inom HCI är hur man hanterar komplexitet så att man kan designa system som är lätta att lära och använda. Ett vanligt sätt att göra detta är att använda sig av kunskap som användaren har inom andra domäner genom att försöka överföra denna till datorvärlden. Detta kan man göra genom att använda sig av metaforer (Carrol et al. 1988). Wozny (1989) skiljer på två olika typer av metaforer, organisatoriska och funktionella. De organisatoriska används för att ge intryck av en plats eller rumslig orientering. Som exempel kan då nämnas skrivbordet på vilket man placerar filer, eller fönster som erbjuder insyn i en liten del av filsystemet. Den andra typen, de funktionella metaforerna, används för att representera en funktion i ett system. Genom att använda t ex klipp och klistra som en metafor förstår användaren enklare hur den ska kunna flytta text mellan två olika dokument. (Wozny 1989)

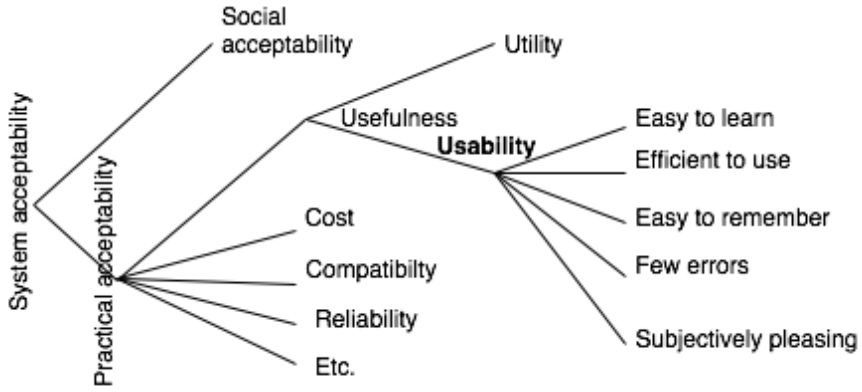
Wozny delar upp metaforers betydelse för användaren i två stadier. I det första stadiet, innan användaren är bekant med systemet, fyller metaforerna en väldigt viktig funktion då de underlättar förståelse för systemet. Allt eftersom användaren blir mer bekant med systemet träder den i det andra stadiet. Användaren skapar sig då en egen mental modell av systemet och kommer att relatera till denna när den ska lära sig nya saker. Denna typ av modell sträcker sig bortom metaforer och har en mer abstrakt betoning. Exempelvis relaterar användaren då inte till hur det fungerar i verkligheten när man klipper och klistrar utan tänker direkt på hur funktionen har fungerat vid tidigare tillfällen Designers bör därför, förutom att tänka på vilka metaforer som kan hjälpa nya användare att förstå, även ta hänsyn till den mentala modell som mer erfarna användare redan har. (Wozny 1989)

Ett annat begreppet som är viktigt att förstå när man talar om användbarhet och interaktionsdesign är konceptuell, eller mental, modell (Norman 1988). Med detta begrepp avser Norman den mentala modell som skapas när vi försöker förklara hur något fungerar för oss själva. Vi skapar en förståelse för hur t ex en funktion i ett system fungerar, denna modell kommer nödvändigtvis inte stämma överens med vad som faktiskt händer. Detta kan leda till problem då användaren tror att en aktion ska utföra en viss funktion medan resultatet blir något helt annat. Norman utvecklar problematiken genom att skilja på tre typer av mentala modeller. Den första, designmodellen är den modell som utvecklaren har, alltså den bild han själv har av systemet. Den andra är den modell som användaren skapar för sig själv. Idealet är naturligtvis att dessa ska vara lika men så är ofta inte fallet. Eftersom utvecklaren endast kan förmedla sin modell genom systemet självt, blir den tredje typen ytterst viktig, den benämns som systemets image. För att användaren ska kunna skapa en korrekt modell är det viktigt att utvecklaren först har en egen modell som är funktionell och användbar för att sedan förmedla denna via systemets image.

3.2 Användbarhet

”En produkts användbarhet visar sig i *samspelet* mellan produkten och dess användare *över en tidsperiod* – kort sagt: i användning” (Ottersten & Berndtsson 2002, s. 14)

Studier kring användbarhet är en del av det ämne som oftast kallas för människa-datorinteraktion [MDI]. Målet med MDI är att producera användbara, säkra och funktionella system (Preece et al. 1994, s. 14). Nielsen (1993b, s. 25) beskriver användbarhet som en del i ett slags generell egenskap som har att göra med hur ett system accepteras. Denna kallar han för *system acceptability* (Figur 3.1). Förutom användbarhet ryms i detta begrepp saker som kostnad, kompabilitet och social acceptans. Han inbegriper alltså även flera saker som inte har med själva interaktionen att göra.



Figur 3.1: System acceptability (Nielsen 1993b, s.25)

Vi kommer inte här att gå närmre in på de övriga objekten, det räcker att notera att de finns, utan vårt fokus ligger på användbarhetsbegreppet. Vad menas då med detta begrepp?

Användbarhetsbegreppet har länge funnits men har gått från att tidigare endast fokusera på det mänskliga systemet till att se produkten och dess användning i ett sammanhang. Det handlar alltså idag om ett begrepp som endast uppstår vid användning, användbarhet är ingen egenskap såsom färg eller funktion (Ottersten & Berndtsson 2002). Att en produkt, i vårt fall ett datorsystem, ska vara användbar känns som en självklarhet. Lika självklart borde det vara att olika produkter kan vara olika användbara, man talar då om att de har olika hög grad av användbarhet. För att kunna tala i dessa termer är det nödvändigt att specificera, eller definiera exakt vad som menas med begreppet. Vilken definition man väljer kommer naturligtvis att påverka en sådan jämförelse och det är därför viktigt att man är tydlig med vad som avses. Vi kommer lite senare att presentera två av de vanligast förekommande definitionerna. Först vill vi dock kort diskutera vad det är som gör att ett fokus på användbarhet är så viktigt.

3.2.1 Fördelar

Ottersten och Berndtsson (2002, s. 21-26) har samlat ett antal av de viktigaste fördelarna som kan nås med ett fokus på användbarhet.

- *Ökad produktivitet.* En produkt med högre grad av användbarhet leder till att den är lättare att arbeta med och därmed tar mindre tid i anspråk.
- *Minskad inlärningstid.* Kort inlärningstid är alltid positivt, särskilt hos produkter som inte används ofta.
- *Minskade kostnader och kortare tid för utveckling.* Om man inte har särskilda tekniker för att nå ett förväntat syfte med en produkt läggs mycket tid till tyckande som sedan kan komma att ändras. Om man istället baserar produktens utformning på kunskap om användbarhet har man större möjlighet att nå rätt från början.
- *Engagemang hos beställare och användare.* Med effektiva tekniker för beställar- och användarmedverkan minskar ökar möjligheten att engagera dessa personer.

- *Minskade livscykelkostnader.* Med ett fokus på användbarhet minskar kostnader för drift, utbildning, support och underhåll. Utslaget på hela livscykeln blir det stora skillnader.
- *Ökad tillfredsställelse för användaren.* Användare upplever större tillfredsställelse eller belåtenhet om produkten uppför sig så som de förväntar sig. Man kan då tänka sig att användarens lust att använda produkten ökar och därigenom kan man även öka produktiviteten.
- *Stärkt varumärke.* Om produkten är starkt förknippat med varumärket så är det naturligtvis viktigt att den inte upplevs som krånglig eller svårarbetad då det kan påverka varumärket och därmed andra produkter.

Vi kan göra ett kort tankeexempel för att illustrera den första punkten, *ökad produktivitet*. Om man tänker sig att alla personer som idag använder datorer i arbetet kunde spendera tio minuter färre genom att slippa strul eller genom effektivare arbete skulle det totalt bli väldigt många arbetstimmar som kan användas till något mer produktivt. Som vi tidigare påpekat använder 65% av Sveriges befolkning datorer dagligen. Dessa tio minuter skulle då totalt motsvara nära 1 000 000 timmar, dagligen. Siffrorna är inte exakta men visar ändå på vikten av att fundera kring användbarhet.

3.2.2 Varför görs det inte?

Om det nu finns så stora fördelar med att fokusera på användbarhet, varför görs det inte alltid? Ottersten och Berndtsson (2002) menar att det beror på tre faktorer; myter, brister i utvecklingsprocessen samt brist på kunskap. När de talar om myter avses felaktiga föreställningar kring användbarhet som existerar i många sammanhang idag. Listan på myter är ganska lång, vi nöjer oss med att redovisa några stycken.

Myten om att användbarhet ökar utvecklingskostnaderna beror på att man tänker sig att aktiviteter för att öka användbarheten ses som tillägg till det nuvarande arbetssättet, snarare än ett nytt arbetssätt. På samma sätt kan det finnas en tanke i linje med ”Det har gått bra hittills så varför ändra?”. En annan myt menar Ottersten och Berndtsson är att det finns en föreställning om att utvecklarna borde känna till alla tänkbara problem eftersom de arbetar med produkten hela tiden. Det man missar när man tänker så, är att användare och utvecklare har olika förhållningssätt till produkten, samt en annan erfarenhet och förförståelse och därför inte uppfattar den på samma sätt. Vidare finns det en syn på att svårigheter med en produkt kan lösas med utbildning, support och dokumentation. Detta är en möjlig lösning, men att föredra är att utveckla en produkt som förklarar sig själv och inte kräver omfattande insatser efter utvecklingen är avslutad.

Den andra punkten, brister i arbetssätt, handlar om att det ofta idag fokuseras för mycket på projekt- och teknisk kvalitet snarare än på hög användningskvalitet. Det arbetas helt enkelt på ett sätt som inte är optimalt för att utveckla produkter med hög grad av användbarhet. Än värre är den tredje punkten, att det helt enkelt saknas kunskap om vad som egentligen menas med begreppet. Alla produkter utvecklas rimligen med en användare i åtanke, problemet är att genom att inte ha kunskap om de processer och tekniker som ligger bakom effektiv utveckling för användbarhet går man miste om väldigt mycket.

3.2.3 Definition

Ovan har vi kort gått igenom bakgrund samt beskrivit vad som generellt brukar räknas in i begreppet användbarhet. Exakt vad som avses är dock inte alltid helt klart. Vi ska nedan ge exempel på två godtagna definitioner.

Den första definitionen, som till exempel används i Gulliksen och Göransson (2002), är från ISO

9241-11 (1998) där man definierar användbarhet som:

> ”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.”

Grunden i definitionen är de tre begreppen *effectiveness, efficiency* och *satisfaction.*

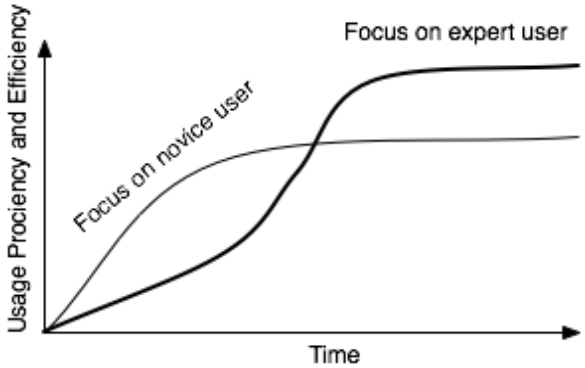
- Med *effectiveness* avses hur väl användaren kan utföra en uppgift. Det är ett strikt resultatinriktat begrepp som inte tar hänsyn till hur uppgiften utförs eller hur lång tid det tar, endast hur väl den utförs.
- Tidsaspekten behandlas istället i begreppet *efficiency.* Hur lång tid tar det att utföra en specifik uppgift?
- Med *satisfaction* menar man hur användaren uppfattar objektet. Föredras ett system framför ett annat? En ytterst subjektiv värdering.

Att definiera begreppet på detta vis har fördelen att det delas upp i mätbara termer och kan därmed användas till att jämföra olika produkter och säga att den ena i en viss situation är mer användbar än den andra. Man poängterar då att användbarhet inte är något absolut begrepp, produkter är inte antingen användbara eller inte, utan man talar om olika hög grad av användbarhet. (Gulliksen & Göransson 2002, s. 62)

Den andra definitionen vi vill redogöra för presenteras av Jakob Nielsen (1993b, s. 26-37). Som vi kan utläsa av Figur 3.1 består användbarhet, enligt Nielsen, av fem delar. Nielsen menar att genom att definiera användbarhet i dessa attribut så skapar man en systematisk approach snarare än ett abstrakt koncept. På detta vis blir det även lättare att utvärdera och förbättra kunskapen kring ämnet.

Learnability (Lärbarhet)
Efficency (Effektivitet i användande)
Memorability (Memorerbarhet)
Errors (Fel)
Satisfaction (Belåtenhet)

Lärbarhet handlar om hur lätt ett system är att lära för användaren. Det första man måste göra när man står inför ett nytt system är att lära sig hur det fungerar, åtminstone de grundläggande funktionerna. Om systemet är lätt att lära underlättar det naturligtvis. Nielsen (1993b, s. 28) illustrerar lärbarheten (Figur 3.2) genom användandet av en graf där effektiviteten visas på y-axeln och tid på x-axeln. Användaren börjar på 0% effektivitet och den ökar sedan med tiden i en s-form. Desto snabbare ökning, desto lättare är systemet att lära. Man bör dock ha i åtanke att alla människor har olika förutsättningar och erfarenheter när det gäller inlärning. En van datoranvändare får en brantare kurva, en användare som uppgraderar från ett tidigare system börjar sin kurva högre upp. Man kan alltså skilja mellan flera typer av lärbarhet.



Figur 3.2: Inlärningskurva för ett hypotetiskt system som visar skillnaden i fokus på expert- och nybörjaranvändare. (Nielsen 1993b, s. 28)

Effektivitet i användande. Detta attribut avser den effektivitet som användaren uppnår när s-kurvan börjar plana ut, det handlar alltså om en utvärdering av erfarna användare. Detta innebär att detta attribut kan vara svårt att mäta i ett tidigt stadium.

Memorerbarhet är tätt förknippat med lärbarhet. Här mäts hur enkelt det är att komma ihåg ett system, och uppnå samma grad av effektivitet, efter att inte använt det under en tid. Man har i och med detta identifierat en tredje grupp förutom nybörjare och experter, nämligen *casual users*.

Fel. För att ett system ska vara användbart är det nödvändigt att undvika att användare gör många och allvarliga fel, fel som inte upptäcks av användaren eller fel som påverkar arbetet negativt. Vissa fel är ofrånkomliga men kan enkelt rättas till direkt av användaren och påverkar då inte effektiviteten.

Belåtenhet behandlar den subjektiva belåtenheten en användare kan nå när denne använder ett system. För vissa typer av produkter är detta viktigare än t ex effektiviteten, såsom hos datorspel eller andra produkter för underhållningssyfte. Nielsen gör också en skillnad mellan attityden användaren känner till produkten ifråga och till datorn i allmänhet. Det senare handlar snarare om det som han kategoriserar som *social acceptability* (se Figur 3.1).

Den stora skillnaden mellan de två definitionerna är Nielsens användning av begreppet lärbarhet samt urskiljningen ISO-definitionen gör mellan *efficiency* och *effectiveness*. Vi tycker att dessa aspekter är viktiga och kommer att ha dem i åtanke under resten av arbetet.

3.2.4 Hur mäts användbarhet?

Vi har nu definierat vad begreppet användbarhet innebär och tänker därmed gå vidare med att klargöra vad för alternativ som finns tillgängliga när det gäller att mäta densamma. Den enklaste formen av mätning är att låta användare värdera en artefakt, utifrån de attribut vi redogjort för ovan. Varje attribut skulle då kunna få ett genomsnittsvärde som kan jämföras med ett tidigare uppsatt mål. Värderingen skulle kunna utföras av en grupp utvalda användare som genomför ett urval av specificerade representativa uppgifter. Det finns dock betydligt mer systematiska metoder. I litteraturen skiljer man traditionellt på två typer av utvärderingsmetoder, analytiska och empiriska, eller inspekterande och testande om man så vill. Inom varje kategori finns ett flertal olika metoder. Holzinger (2005) sammanfattar de vanligaste metoderna och dess egenskaper i Tabell 3.1.

Tabell 3.1: Sammanställning av olika mätmetoder

	Inspection Methods			**Test Methods**		
	Heuristic Evaluation	Cognitive Walkthrough	Action Analysis	Thinking Aloud	Field Observation	Questionnaires
Applicably in Phase	all	all	design	design	final testing	all
Required Time	low	medium	high	high	medium	low
Needed Users	none	none	none	3+	20+	30+
Required Evaluators	3+	3+	1-2	1	1+	1
Required Equipment	low	low	low	high	medium	low
Required Expertise	medium	high	high	medium	high	low
Intrusive	no	no	no	yes	yes	no

I tabellen skiljer Holzinger på i vilken fas metoden är tillämpbar, hur lång tid den tar att utföra, hur många användare och utvärderare som behövs, vilken kunskap och utrustning som krävs, samt huruvida det är en metod som är inkräktande. Ur tabellen går det att utläsa att den heuristiska analysen är möjlig att genomföra i alla faser av en utveckling.

Av dessa olika alternativ har vi valt att se närmare på den heuristiska utvärderingen. Som vi kan utläsa av tabellen kan den användas i alla faser av en utveckling, den kräver mindre tid och resurser och inga användare. Detta gör att den är lämpad för den typ av studie vi ämnar genomföra.

3.3 Heuristic evaluation

3.3.1 Bakgrund

Heuristisk utvärdering är en metod för användbarhetsutvärdering som utvecklades av Jakob Nielsen och hans kollegor. Metoden går kortfattat ut på att experter går igenom och utvärderar ett gränssnitt utifrån ett antal på förhand givna principer, *heuristics*. Tanken är då att bra design följer vissa generella principer, och genom att testa hur väl ett gränssnitt stämmer överens med dessa kan man göra en bedömning av användbarheten (Faulkner 2000, s. 179). Hur många, och vilka principerna bör vara är upp till utvärderarna att fastställa, utifrån relevant litteratur och riktlinjer, beroende på vilken produkt det är som ska testas (Preeceet al. 2002). Det är dock viktigt för utvärderaren att kunna bedöma relevansen hos dessa principer. Det finns nämligen principer som finns kvar genom historiska olyckshändelser, och andra kanske inte längre är giltiga. Eftersom hela utvärderingen baseras kring dessa principer är det centralt med ett kritiskt förhållande till dessa (Faulkner 2000, s. 179). Vi kommer senare i kapitlet att redogöra för ett par av dessa set, eller samlingar, av principer som kan användas för denna typ av utvärdering.

3.3.2 Hur

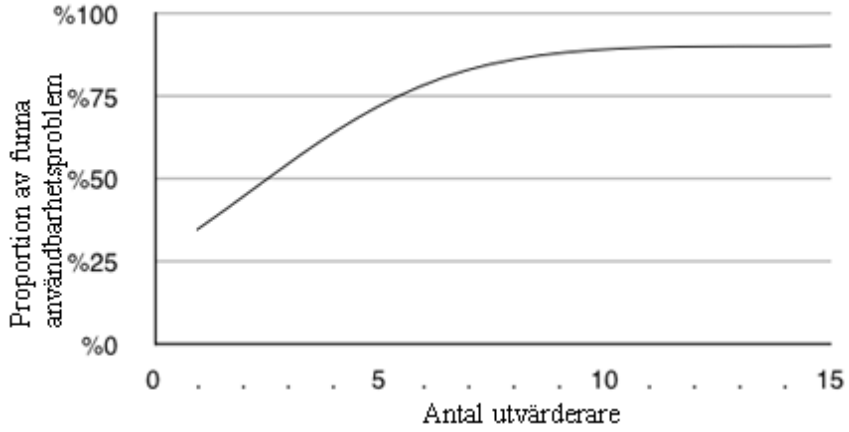
Nielsen (1993, s. 155) beskriver metoden som att ”Heuristic evaluation is done by looking at an interface and trying to come up with an opinion about what is good and bad about the interface”. Denna typ av utvärdering kan antas att användare utför ganska frekvent, dock utifrån en lista baserad på intuition och sunt förnuft snarare än fördefinierade principer. Det Nielsen förespråkar är istället en systematisk inspektionsmetod för att finna användbarhetsproblem hos ett gränssnitt.

Utvärderingen går till så att utvärderaren går igenom gränssnittet ett flertal gånger och inspekterar olika situationer och jämför de med den lista av principer som fastslagits. Till detta kommer utvärderarens egna kunskap och erfarenheter kring användbarhet. Varje utvärderare ska gå igenom systemet ensam, endast när de fullföljt uppgiften får de träffas för att diskutera och samla sina resultat. En session tar i regel en till två timmar. Resultatet av utvärderingen är en lista av användbarhetsproblem som refererar till den eller de principer som de bryter. Analysen resulterar inte i tänkbara lösningar på problemen, men eftersom beskrivningen utgår ifrån kända användbarhetsprinciper blir det ofta lätt att generera en lösning. Om man vill inkludera en problemlösning kan man avsluta sessionen med en debriefing av brainstormingtyp, där de olika utvärderarna och representanter från de ansvariga för designen deltar. (Nielsen 1993a)

3.3.3 Av vem

Så som analysen beskrivs av (Nielsen (1993b) genomförs den av en eller flera experter under olika faser av ett utvecklingsarbete. Det är viktigt att poängtera att utvärderaren alltså inte går igenom systemet och registrera var han har problem, istället har experten på området i uppgift att förutse var användarna kan tänkas få problem. Angående hur många utvärderare som bör delta har Nielsen i en studie visat att en ensam person endast fann ca 35% av problemen, och eftersom olika utvärderare tenderar att finna olika typer av problem, är det viktigt att låta flera personer genomföra analysen. Hur många utvärderare som anlitas är beroende av kostnadsfaktorer, det kan vara gynnsamt att låta en ytterligare person analysera gränssnittet men detta är inte alltid kostnadseffektivt. En bedömning får göras om det är värt kostnaden för en extra utvärderare för möjligheten att finna något extra problem som de övriga inte upptäckt.

Som vi ser av Figur 3.3 ökar antalet funna problem ganska kraftigt med antalet utvärderare upp till ungefär tio stycken, därefter avtar effekten. Nielsen redogör också för en studie som visar hur stor betydelse expertens kunskap har. Tre grupper delades in i personer som endast hade datorkunskaper, personer som hade kunskaper i användbarhet men som inte var specialiserade på gränssnitt, samt en tredje grupp som hade kunskaper i både användbarhet och gränssnitt. Studien visade då att den första gruppen fann 22%, den andra 41% och den tredje 60% av användbarhetsproblemen. Experterna på området fann alltså 1,5 gånger så många problem som de utan kunskaper i området. Detta visar att det är bättre att ha personer med stor kunskap, men även att icke-utbildade kan finna ett antal problematiska områden. (Nielsen 1993b, s. 161)



Figur 3.3: Funna problem som funktion av antal utvärderare (Nielsen 1993b, s. 156)

3.3.4 För- och nackdelar

Det finns flera fördelar med den heuristiska utvärderingen. Dess främsta fördel är också den med störst begränsning och innefattar uteslutandet användare. Detta gör att det blir en väldigt flexibel metod som inte är begränsad av en prototyp för testning vilket gör att den är tillämpbar även på idéstadiet, innan produkten ens har börjat ta form. Det innebär också att den inte är särskilt tidskrävande om man jämför med testning som inkluderar användare. Den uppenbara nackdelen är att man blir beroende av experterna och deras kunskap. Som vi beskrev ovan skiljer sig resultatet från utvärderingen markant beroende på expertemas bakgrund och erfarenhet inom området. Vidare måste man vara medveten om begränsningen att man på intet sätt kommer att finna alla problem genom denna metod. Det bästa vore naturligtvis att använda sig av både heuristisk utvärdering och någon form av användartest men detta skulle också bli kostsamt.

Användartester har problem med reliabiliteten eftersom användare har stora individuella skillnader (Nielsen 1993b, s. 166). Reliabilitet bestäms utefter grundligheten i datainsamlingen, för att uppnå en god reliabilitet i denna skulle ett stort antal användare behöva testa produkten.

3.4 Olika heuristics

Nielsen (1993b) presenterar i samband med hans redogörelse för metoden ett set av principer som han menade var lämpliga att utgå ifrån. Dessa har blivit allmänt accepterade och är vanligt förekommande i litteraturen. Det finns dock set som andra författare presenterat som skiljer sig något åt, även om de ofta är förslående lika. Det tycks alltså råda ett visst konsensus kring vad som är god design, vilket gör de få skillnader som finns än mer intressanta. Vi kommer nedan att redogöra för några av de mest populära samlingarna för att ge oss en bredare bas för vår analys.

3.4.1 Nielsens principer

Nielsen (1993b) tar upp tio fundamentala principer som han menar bör ingå i analysen. Dessa har härletts ur ett omfattande empiriskt material och är generellt accepterade i litteraturen. Vi väljer att använda Ottersten och Berndtssons (2002) översättningar av samma begrepp i vår redogörelse.

1. Enkel och naturlig dialog.
2. Använd ett naturligt språk.
3. Minimera användarens minnesbelastning.
4. Enhetlighet.
5. Förse användaren med återkoppling.
6. Förse användaren med klart markerade funktioner för att avbryta dialogen.
7. Effektiv användning.
8. Preciserade felmeddelanden.
9. Förhindra fel.
10. Hjälp och dokumentation.

Enkel och naturlig dialog. Ett gränssnitt bör vara så enkelt som möjligt, eftersom varje extra funktion eller information är en ytterligare sak att lära sig, behöva leta igenom, eller missuppfattas. Vidare bör systemet matcha användarens uppgift på ett så naturligt sätt som möjligt. Norman (1988) talar om mapping, när han avser förhållandet mellan användarens uppfattning av något och hur detta representeras i systemet. Idealet är att ge användaren precis den informationen han behöver, och inte mer, vid precis rätt plats och tillfälle.

Angående den grafiska layouten skriver Nielsen att den bör använda sig av den mänskliga perceptionen som en grund. Detta inkluderar hur saker och ting bör vara grupperade, hur information lättast uppfattas mm. Vidare diskuterar han vikten av valet av färger och att det viktigaste är att inte använda dessa för att sända information. Många individer har defekt färgseende, därför bör man inte begränsa sig till just färger för information. I övrigt skriver han att vinsterna av att välja optimala kombinationer endast är små så länge man undviker de värsta exemplen, så som gul text på blå bakgrund och liknande.

Använd ett naturligt språk. All information bör presenteras i termer och språk som är naturligt för användaren. Man bör också utgå ifrån användarens perspektiv i dialoger, d v s ”du har utfört x” istället för ”vi har utfört x åt dig”. Systemet bör heller inte ha några restriktioner, såsom antal tecken eller specialtecken, då det gäller namngivning av filer eller dylikt.

Under denna punkt behandlas även mapping och metaforer. Mapping beskrivs som förhållandet mellan en användares konceptuella bild av något och hur detta representeras i systemet. Ett sätt att hantera detta är genom användandet av metaforer. Metaforer låter användaren förstå en företeelse i systemet utifrån en förståelse av ett ting i den verkliga världen. Man måste dock vara försiktig så att man inte med en metafor antyder att systemet klarar av mer än det egentligen gör, eller att en metafor begränsar användaren. T ex innebär uppfattningen att en ordbehandlare fungerar som en skrivmaskin att användaren går miste om många viktiga funktioner.

Minimera användarens minnesbelastning. Systemet bör generera valbara alternativ till olika dialogelement. Vidare bör kommandon och funktioner synliggöras i t ex menyer för att lätt kunnas nå. När systemet vill ha input bör det beskriva i vilken form detta ska ske och ge ett tänkbart förslag. Det är lättare för användaren att modifiera något än att skapa det från grunden.

Datorer är bättre på att komma ihåg saker än användare. Systemet bör därför baseras på ett fåtal regler eller instruktioner som är lättillgängliga och konsekvent applicerbara genom hela produkten. T ex bör då kommandot ”copy” ha samma funktion oavsett i vilken del av systemet man arbetar.

Detta leder vidare in på *enhetlighet.* Enhetlighet är en av de mest grundläggande användbarhetsprinciperna och kan appliceras på det mesta. Liknande information bör presenteras på liknande sätt och plats, samma kommando bör göra samma sak osv. I den mån det finns färdiga standarder, som för t ex dialogrutor, bör dessa följas.

Förse användaren med återkoppling. Systemet ska fortlöpande informera användaren vad det arbetar med och hur det tolkar användarens input. Återkoppling är särskilt viktigt när systemet har lång responstid.

Förse användaren med klart markerade funktioner för att avbryta dialogen. För att öka användarens känsla av kontroll i en dialogsituation bör det hela tiden finnas lätta sätt att avbryta dialogen. Det bör därför t ex finnas en ångra-funktion lätt tillgänglig i hela systemet.

Effektiv användning. Ett gränssnitt måste fungera för människor med olika erfarenheter. Det är därför viktigt att det är användbart utifrån kunskap av endast ett fåtal enkla regler. Det är dock också viktigt att erbjuda alternativ för den mer erfarne användaren att effektivisera sitt bruk. Detta görs enkelt t ex genom implementerandet av kortkommandon och genvägar.

Preciserade felmeddelanden. Felmeddelanden är viktiga ur användbarhetssynpunkt dels som hjälp att komma ur en situation där ett fel har uppstått, dels för att skapa bättre förståelse för systemet och för att lära användaren hur denne bör göra i framtiden. Meddelandena bör vara formulerade med ett klart språk snarare än obskyr kod. De ska därmed kunna förstås utan att användaren behöver hjälp av en extern manual eller dylikt. De bör vidare vara precisa samt erbjuda konstruktiv hjälp för hur användaren kan lösa problemet. Slutligen bör de vara artigt formulerade och inte skuldbelägga användaren.

Förhindra fel. Ännu bättre än att ha bra felmeddelanden är naturligtvis att förebygga så att fel inte uppstår. Det finns flera exempel på situationer där fel är särskilt benägna att dyka upp, som enkelt hade kunnat undvikas. Det kan röra sig om att erbjuda en lista av alternativ istället för att låta en användare manuellt skriva in kommandon, eller att ha dialoger som frågar om användaren verkligen är säker på att denne vill genomföra en handling.

Hjälp och dokumentation. Idealet är naturligtvis att ett system är så enkelt att det förklarar sig själv. Så är dock oftast inte fallet. Därför är det nödvändigt med bra hjälp och dokumentation. Man bör också tänka på att användare oftast inte vill läsa manualer förrän det är nödvändigt. De vill hellre

börja använda systemet direkt och det är därför bra med hjälp som kan fås direkt i systemet snarare än en utskriven papperskopia.

3.4.2 First principles of interaction design

Nielsens heuristiska utvärdering påminner i mångt och mycket om generella designprinciper som ofta förespråkas. En av de mer inflytelserika forskarna på området, Bruce Tognazzini (2003), har en egen lista med denna typ av principer, kallad first principles of interaction design. Han menar att dessa principer är generellt giltiga för alla gränssnitt, oavsett om det gäller för en webbshop eller för en traditionell applikation. Tognazzini tar upp ett flertal principer, av utrymmesskäl går vi dock inte igenom alla lika noggrant. Detta urval baseras på vilka som bedöms vara särskilt intressanta för vårt ämne eller vilka som skiljer sig från övriga författares. Alla principer beskrivs dock i korthet.

Anticipation (Förutsägande). Ett program bör försöka förutsäga vad en användare vill göra och behöver. Man ska inte förvänta sig att en användare söker efter information, det är bättre att erbjuda den och alla nödvändiga verktyg som behövs för en process.

Autonomy (Självständighet). Gränssnittet och hela datorn tillhör användaren. Dock behövs det vissa regler som reglerar användningen, detta för att användaren ska känna sig säker och ha kontroll. Dessa regler måste dock hela tiden tydliggöras och information måste ges som visar vad som händer.

Color blindness (Färgblindhet). Varje gång man använder färg för att ge information måste man också vara noggrann med att erbjuda informationen på ett sekundärt sett. En betydande del av befolkningen har någon form av defekt färgseende.

Consistency (Konsekvens). Tognazzini förespråkar en lista i vilken ordning olika typer av konsekvens bör prioriteras. För att meningen inte ska gå förlorad väljer vi att inte översätta begreppen.

1. Interpretation of user behavior, e. g., shortcut keys maintain their meanings.
2. Invisible structures.
3. Small visible structures.
4. The overall "look" of a single application or service-splash screens, design elements.
5. A suite of products.
6. In-house consistency.
7. Platform-consistency.

Defaults. Här redogör Tognazzini för hur förvalda värden bör hanteras.

Efficiency of the User (Användarens effektivitet). Tognazzini har ett flertal punkter som har med effektivitet att göra.

- Se till användarens effektivitet, inte datorns. Arbetskraft är betydligt dyrare än maskinerna. Det centrala är därför inte vad datorerna kan göra, utan vad människorna kan utvinna ur datorernas system, ett fokus på att bara göra datorer snabbare är alltså inte nödvändigtvis produktivt.
- Vidare bör man därför se till att användaren hela tiden kan hålla sig sysselsatt, varje gång användaren måste vänta på att datorns respons är förlorad tid.
- Man bör också se längre än till den enskilde individen. På ett stort företag bör man välja en lösning som använder minst arbetskraft totalt, t ex med olika nätverkslösningar.
- De största effektivitetsvinsterna görs i den grundläggande arkitekturen, inte i det enskilda gränssnittet.

Explorable Interfaces (Utforskningsbara gränssnitt). Här skriver Tognazzini om känslan användaren bör få när denne använder sig av ett gränssnitt. Han liknar det vid att erbjuda användaren tydliga

vägar och landmärken, för att sedan låta denne fritt utforska systemet. Detta bör dock kompletteras med ett alternativ som erbjuder minsta möjliga hinder för den användare som vill få jobbet klart så fort som möjligt. För att användaren ska våga utforska systemet är det viktigt att man kan återställa alla ändringar.

Fitts' Law. Tiden att nå ett mål är en funktion av avståndet och storleken på målet. Använd stora objekt för viktiga funktioner, men även kanterna och framförallt hörnen då de är lättast att nå. Här menas helt enkelt att viktiga funktioner, eller funktioner som utförs ofta, bör placeras lättillgängliga.

Human-Interface Objects (Mänskliga gränssnittsobjekt). Här tas förhållandet mellan objekt i verkligheten och objekt i gränssnittet upp.

Latency Reduction (Reducera latens). Man bör minska tiden som systemet är upptaget och användaren inte kan använda det genom multi-threading. Vidare bör användaren ges tydlig feedback om hur länge systemet kommer vara upptaget, samt när det är ledigt igen, genom olika signaler. Naturligtvis bör man också sträva efter att förbättra snabbheten, bl a genom att ta bort element som inte har någon funktion.

Learnability (Lärbarhet). Idealet är att alla produkter skulle gå att använda fullt ut med en gång, så är dock inte fallet. Alla produkter har istället en inlärningskurva. Det man bör ha i åtanke är att begränsa kohandeln mellan användbarhet och lättlärdhet. Allt för enkla system är inte alltid särskilt användbara då de har begränsad effektivitet.

Metaphors (Metaforer). Använd metaforer som gör att användarna genast uppfattar de minsta detaljerna av den konceptuella modellen. Gör metaforerna levande genom att tilltala olika sinnen likaväl som att hänvisa till användarens minne.

Protect the User's Work (Skydda användarens arbete). Se till så att användaren aldrig förlorar sitt arbete p g a något fel de utför.

Readability (Läsbarhet). Denna punkt behandlar hur text bäst läses, storlek, typsnitt och dylikt.

Track State (Registrera status). Vad användaren har gjort, och var denne befunnit sig bör sparas i en fil så att arbetet kan återupptas vid samma position nästa gång.

Visible Navigation (Synlig navigering). Erbjud användaren illusionen att användaren alltid är på samma ställe och informationen förs till dem. Detta i motsats till när man använder Internet och surfar runt utan att riktigt veta var man befinner sig. Detta erbjuder, förutom att det minskar behovet av mentala kartor, även större känsla av kontroll och autonomi för användaren.

3.4.3 Norman's seven principles for transforming difficult tasks into simple ones

Normans (1988) sju principer är avsedda att användas i designfasen, det rör sig alltså om principer man bör använda för god design mer än principer som är till för utvärdering. Faulkner (2000) menar dock att de mycket väl kan användas till detta. De behandlar ett par intressanta aspekter vilket gjort att vi valt att diskutera dem i vår genomgång.

1. *Use both knowledge in the world and knowledge in the head.* Norman skiljer på kunskap i världen och kunskap i huvudet hos individen. Man bör använda sig av kunskap som redan finns i världen, som därmed inte behöver kommas ihåg då man hela tiden kan få fram den.

Kunskap i huvudet är av typen man måste lära sig, för att sedan ta fram ur minnet nästa gång den behöver användas. Kunskap i världen är att föredra i de fall då det går att översätta den på ett logiskt och lättolkat sätt.

2. *Simplify the structure of the task.* Uppgifter bör vara enkla så att de minimerar planerandet som krävs för att lösa dem. Komplexa uppgifter kan omstruktureras så de blir enklare, t ex med hjälp av ny teknologi.
3. *Make things visible. Bridge the gulfs of execution and evaluation.* Designers bör göra det uppenbart vad som kan utföras och vilken effekt olika handlingar har. Det som sker måste stämma överens med det som användaren avser att göra. Detta beskriver Norman som att man ska bygga över klyftan av utförande och utvärderande.
4. *Get the mappings right.* Det är viktigt att försäkra sig om att användaren förstår sambandet mellan intentioner och möjliga handlingar, mellan olika handlingar och dess effekt på systemet, systemets faktiska tillstånd och det som syns eller hörs, samt mellan systemets tillstånd och vad som förväntas av användaren.
5. *Exploit the power of constraints both natural and artificial.* Använd begränsningar så att användaren känner att det bara finns en möjlig väg, den rätta. En diskett går t.ex. bara att föra in på ett håll.
6. *Design for error.* Antag att varje fel som kan göras kommer att göras och förebygg dem. Se vidare till så att användaren hela tiden har en utväg, ett sätt att återställa om något fel inträffar.
7. *When all else fails, standardize.* När det inte finns något logiskt sätt att utföra en design, bör den standardiseras så att den genomgående utförs på samma sätt.

3.5 Applicerandet i vårt fall?

Heuristisk utvärdering är en etablerad metod som används i olika stadier av en utvecklingsprocess. Vi ämnar undersöka ett väldigt stort och avancerat system utifrån liknande principer. Vi kommer dock att utforska utifrån en något annorlunda nivå. Operativsystem är oerhört komplexa system och en heltäckande analys av dessa är inte möjlig att genomföra inom våra tidsramar. Vi har alltså inte för avsikt att påvisa alla problem som finns med Windows. Vi menar dock att de principer som används i den heuristiska analysen kan appliceras för att påvisa några av dem, samt att se hur alternativa system löser samma uppgifter. Det rör sig alltså om en modifikation av den ursprungliga metoden.

4 WIMP

4.1 Designprinciperna

Vi kommer nedan att redogöra för ett antal principer som vi menar karaktäriserar WIMP. Principerna är hämtade främst från Apples (2006) egna riktlinjer för gränssnittsutveckling. Detta dokument har haft i princip samma utformning och innehåll sedan det först lanserades i början på 90-talet och beskriver ingående ett antal punkter och företagets syn på dessa. De är utformade så, att man menar att för att utveckla ett bra och användbart gränssnitt bör man i så stor utsträckning som möjligt ta hänsyn till var och en av dessa. Apple var som vi tidigare diskuterade, det första företaget som utformade ett WIMP-baserat gränssnitt i en större skala. Detta faktum, tillsammans med dokumentets stabilitet över tiden, gör att vi anser att det är lämpligt som utgångspunkt för vilken innebörd vi lägger i begreppet WIMP. Vi har ändock, valt att komplettera detta, på ställen vi ansett det nödvändigt, med information från två andra utvecklare av WIMP-baserade gränssnitt, GNOME (2004) och Microsoft Cooperation (2007).

4.1.1 Metaforer

En metafor är ett bildligt uttryck för något som kan verka komplext och kan ibland underlätta förståelsen för det man försöker beskriva. Syftet med metaforer i användargränssnitt, är att skapa en motsvarighet till verkliga föremål, eller en förenklad bild av kontexten, i gränssnittet. Man vill utnyttja redan väl igenkända föremål, som folk sedan tidigare lärt sig hur de fungerar.

Apple (2006) menar att man ska ta till vara på användarens kunskap om omvärlden, genom att implementera metaforer som underlättar förståelsen över systemets funktioner. Man bör dock bara använda metaforer som användare redan är familjära med och göra dem så pass uppenbara att användaren inte kan missförstå dem.

4.1.2 Konceptuell-/Mental modell

Användaren har en mental modell, eller en syn, på vilka uppgifter ett system utför, redan innan hon eller han arbetat med systemet. Denna modell, eller syn, beskrivs av Apple som *Reflect the user's Mental Model* (Apple, 2006, s. 40). Modellen beror på kunskap från tidigare system eller erfarenhet från verkligheten. Som exempel har vi en verklig kunskap om hur man skriver och skickar brev med post. De flesta datoranvändare har även använt system för att skicka e-post via Internet. Baserat på detta har användaren en konceptuell bild på hur det ska gå till när man skriver och skickar brev, så som skapa ett nytt brev, välja mottagare för att sedan skicka brevet. Apple menar att ett system som bortser från användarens förväntningar, blir svårt att använda och ej tilltalande.

4.1.3 Direktmanipulation

Grundidén med direktmanipulation i ett gränssnitt är att varje objekt visualiseras grafiskt för användaren, och där användaren direkt manipulerar objektet. Ett exempel är, återigen,

skrivbordsmetaforen. Vi återkommer gärna till Apples Macintosh, där man drar dokument till papperskorgen för "kasta" det. Ett klassiskt exempel på direktmanipulation. Andra exempel kan vara möjligheten att ändra storlek på ett grafiskt objekt, drag-and-drop och flytta markerad text från ett textdokument till ett annat o s v. Apple skriver, med avseende på detta begrepp, att det är viktigt att stödja det när användaren förväntar sig det.

4.1.4 Användarkontroll

Målet med användarkontroll, enligt Apple, är att tillåta så mycket kontroll som användaren av systemet klarar av att inneha. Det är t ex inte lämpligt att låta användaren få full tillgång till avancerade funktioner och inställningar i ett system, om man vet med sig att användaren är en nybörjare. Man vill undvika att användaren utför handlingar som kan skada systemet. Då kan varningar vara bra för att informera om konsekvenser eller för att användaren utfört något av en olyckshändelse, men det är viktigt att man låter användaren fortsätta om denne så önskar.

4.1.5 Återkoppling och kommunikation

En annan viktig princip för god användbarhet är, enligt Apple att man låter användaren få viktig information om vad som händer med systemet presenterat för sig. Om användaren anropar ett kommando eller utför ett anrop på ett objekt, ska detta medföra någon typ av respons till användaren. Apple nämner dialogrutor, som ett sätt att ge feedback. Vanligen möts man av en dialogruta när man vill stänga ett fönster, innehållandes, ett dokument, med alternativ om man vill spara dokumentet eller inte.

GNOME beskriver detta som "Keep the User Informed" (håll användaren informerad) och bygger på samma princip, att alltid låta användaren få ta del av information, om vad som händer med systemet. Detta bör ske vid ett lämpligt tillfälle. Syftet med *feedback*, enligt principerna för god användbarhet, utformade av GNOME, är att användaren aldrig ska behöva gissa sig till vad som händer och vilken status systemet eller applikationen har.

4.1.6 Konsekvent

Man skiljer på olika typer av konsekvens. Dels ska systemet vara konsekvent med sig självt, d v s, funktioner ska vara lika på olika ställen, dels ska det vara konsekvent med tidigare versioner, och slutligen ska det vara konsekvent med användarnas förväntningar, skriver Apple. Ett konsekvent utseende är viktigt för att användaren ska känna igen sig i olika program. Enligt GNOME är detta den viktigaste principen när man designar gränssnitt i WIMP. Både gränssnitt och uppförande, så som funktioner och dialogrutor bör uppföra sig på ett snarlikt sätt. Ett exempel kan vara när man fyller i ett formulär och när man är klar, trycker man på "Enter" och all ifylld information försvinner. I vanliga fall betyder "Enter" att man är klar och vill gå vidare, men i ett inkonsekvent fall betyder det rensa fält. Syftet med konsekvent design är att användaren får nytta av tidigare erfarenheter från datormiljöer och system, för att förstå ett nytt system.

Att möta allas förväntningar är den största utmaningen i ett konsekvent program. Apple skriver att man kan väga detta mot användarens behov av finesser, för att hålla ett program konsekvent.

4.1.7 WYSIWYG

What You See Is What You Get [WYSIWYG] beskriver Apple som att det du ser, är det du har tillgång till. Speciellt viktigt är detta i program där användaren kan manipulera data, så som

webbpublicering och bränningsprogram för CD/DVD. I sådana här fall är det viktigt att det användaren ser på skärmen, är det som återfinns i det slutliga resultatet. Exempelvis förväntar sig användaren att det hon eller han skrivit på datorn, ser likadant ut när det skrivs ut i pappersform. Vid fall där utskriften inte kan, av t ex. tekniska skäl, se likadan ut, är det en god idé att låta användaren förhandsgranska resultatet, så som det kommer att se ut.
Det handlar även, för Apples del om att man inte ska gömma funktioner för användaren. De funktioner som finns tillgängliga ska vara lättåtkomliga och gärna synliga för användaren utan att behöva utforska menyer eller underordnade fönster.

4.1.8 Förlåt användaren

Ett bra sätt att uppmuntra användare att utforska och upptäcka funktioner i ett program, menar Apple, är att tillåta användaren att göra misstag i systemet. Man kan förlåta användaren på flera sätt, ett är att låta användaren gå tillbaka till det tillstånd som rådde innan användaren utförde sin handling. Enligt Apple är det viktigt att användaren kan känna sig trygg i systemet genom möjligheten att gå tillbaka till tidigare tillstånd. Exempel på kommandon som användare känner igen är "Undo" och "Revert to saved", vilka gör att användaren känner sig komfortabel och vill lära sig programmet.

4.1.9 Uppfattad Stabilitet

För att skapa en visuell stabilitet gentemot användaren, menar Apple att operativsystemet ska innehålla återkommando element. Med detta menas att element, så som fönster och ikoner på skrivbordet ska ha samma position hela tiden om användaren inte väljer att flytta dem. Även om man stänger ett fönster, så ska detta ha samma plats när man öppnar samma fönster igen. Detta för att användaren ska kunna förvänta sig att gränssnittet uppträder på ett visst sätt, och där arbetsmiljön är lätt att kännas igen. Apple menar att användaren bör mötas med ett begränsat antal av objekt och funktioner för att manipulera dessa objekt.

4.1.10 Lägeslöshet

Modelessness är ett svåröversatt begrepp, vi använder lägeslöshet. Det syftar på frånvaron av olika tillstånd eller lägen som datorn kan vara i, d v s att datorns beteende förändras beroende på funktion. En typ av *mode,* är när man håller in shift-tangenten så att tangentbordet får en annan funktion. Meningen med denna riktlinje är att som användare ska man kunna utföra operationer när och var som helst i gränssnittet. Enligt Apple ska man alltså inte stänga funktioner bara för att gränssnittet är i ett speciellt tillstånd. D v s det man ska tänka på är att användare alltid ska ha tillgång till samtliga funktioner och inte behöva återgå till föregående fönster eller dylikt för att komma åt en viss funktion. Med andra ord, om flera fönster är öppna så ska alla funktioner vara tillgängliga i respektive fönster.

5 WIMP och användbarhet?

Kapitlet inleds med en problematisering av var och en av de riktlinjer för WIMP som diskuterades ovan. Vi avslutar med en kort diskussion och en sammanfattning av de allvarligaste problemen.

5.1 Problematisering av riktlinjerna

5.1.1 Metaforer

Användandet av metaforer är kanske den mest intressanta av de riktlinjer vi gått igenom. Detta då alla system som bygger på ett WIMP-baserat gränssnitt också har en metafor i grunden. I de fall vi har studerat är denna det vi kallat för skrivbordsmetaforen. Vidare använder man sig inom denna metafor av flera andra metaforer. Man problematiserar dock inte användandet av metaforer mer än att diskutera att de bör vara genomtänkta och reflektera den mentala bild av företeelsen (Apple 2006). Som vi redogjorde för ovan kan det dock finnas nackdelar med detta. En allt för stark koppling till det som metaforen avspeglar kan innebära att man tillskriver objektet egenskaper det inte har eller att man missar vissa egenskaper. Ett annat problem har att göra med effektiviteten. Flera av de principer som ingår i den heuristiska utvärderingen talar om att man bör göra det så effektivt som möjligt för användaren utan att förlora i användbarhet. Frågan är om vi idag verkligen har ett så stort behov av metaforer. Vi har idag en helt annan kunskap än när vi för första gången skulle försöka använda den. Detta är inget som man diskuterar i de riktlinjer vi tagit del av. Här kan vi dra paralleller till när Nielsen talar om olika typer av lärbarhet. Idag är det väldigt få som börjar på 0% effektivitet och behoven av lättförstådda system ser därför annorlunda ut.

Ett annat problem är att skrivbordsmetaforen idag kan tyckas väl begränsad. Den utvecklades under en tid då datorn sågs som ett hjälpmedel för olika kontorssysslor. Den fungerade främst som räknemaskin och ordbehandlare, samt höll ordning på stora mängder filer. Metaforen med datorn som ett skrivbord var då väldigt naturlig. Idag lever vi ett annat samhälle, vi använder vår dator till andra saker. Det första man tänker på när man problematiserar denna metafor är hur den egentligen hanterar nätverk. Detta representeras de facto inte på något sätt alls. Stora företagsnätverk visas fortfarande upp som enskilda skrivbord på varje dator. När man söker information på Internet gör man det genom att öppna ett program som visar en metafor för ett papper, visserligen med länkar men ändock ett papper.

Detta för oss in på nästa problem med de metaforer som används idag. Dessa pappersrepresentationer, på Internet eller nyskapade av en användare, sparas som filer och placeras i mappar. Filerna och mapparna var ett bra sätt att få användare att förstå och hantera sina dokument. Dock kan man fundera på om det verkligen är det mest optimala. Hur många har ett arkivskåp med mappar idag? Varför ska då datorn representera dessa? Vi kommer i de följande kapitlen att visa att det finns flera nytänkande idéer på området.

5.1.2 Konceptuell modell

I de riktlinjer vi ställt samman har man en syn på konceptuella modeller som stämmer väl överens

med de som förs fram i litteraturen. Problemet ligger snarast i att användarnas konceptuella modell idag är starkt påverkad av system som utvecklades för många år sedan. Detta var en tid då man var mycket mer beroende av det som Norman (1988) beskriver som kunskap i världen. Man var tvungen att relatera systemets funktioner till saker som fanns ute i världen. Man fastslår i riktlinjerna att man bör ta hänsyn till den modell som användaren har av vad ett system utför för uppgifter och hur det fungerar. Man måste då idag, vid utvecklandet av system och program leva upp till den konceptuella modell som användare har av tidigare användning av datorer. Man kan då säga att man ser mer till vad användarna efterfrågar, då de inte känner till något annat, än till att utveckla produkter som skulle kunna vara mer användbara.

5.1.3 Direkt manipulation

Med direkt manipulation, krävs det ofta att man använder pekdonet för att komma åt ett visst objekt och manipulera det. Tänk dig en bild som ska konverteras i storlek: I en skrivbordsmiljö måste vi dubbelklicka på objektet (ikonen för bilden) och använda ett tredjepartsprogram för att utföra ytterligare steg för att förminska bilden. I slutändan blir det en lång serie med direkta manipuleringar och det blir inte en så effektiv användning som man bör sträva efter (Nielsen (1993b), men som man förvisso kan lösa med scriptning. WIMP i sig bygger inte på att man kan utföra en serie med manipulationer, men det finns lösningar i många operativsystem, så som Mac OS X, där man valt en teknologi som Apple kallar för Automator (Apple 2007a).

5.1.4 Användarkontroll

Apple menar lite vagt, som vi tidigare tagit upp, att användaren bör få så mycket kontroll som den klarar av. Detta är naturligtvis ett problem då användare kan vara så olika. Vidare går det emot det som Tognazzini (2003) menar när han talar om utforskbara system. Det är viktigt att användaren fritt utifrån vissa regler får utforska systemet. Man bör även då se till att så få handlingar som möjligt är oåterkalleliga.

Å andra sidan kan det vara onödigt att ge användaren för mycket kontroll. Det finns idag många enformiga handlingar som systemet skulle kunna utföra men som idag användaren själv måste utföra, vilket kan tänkas till viss del beror på att man vill att användaren ska ha så stor kontroll som möjligt. Det finns en del exempel såsom automatisk sparning eller automatiskt kompletterande av ord som dykt upp under senare år men det borde finnas många fler av denna sort.

5.1.5 Återkoppling och kommunikation

Återigen är det en punkt där litteraturen och riktlinjerna ligger varandra nära. Man kan dock ana att både Nielsen och Tognazzini vill ta begreppet ett steg längre. Man poängterar att det är av yttersta vikt att användaren hela tiden informeras om vilket tillstånd systemet befinner sig i, mycket frustration kan undvikas om det funnits ett system för att erbjuda olika typer av signaler beroende på hur länge systemet kan tänkas vara upptaget. Man kan även dra kopplingar till Normans diskussion om att överbrygga klyftan för utförande och utvärdering . Det gäller att hela tiden ge användaren den information han kan förvänta sig.

5.1.6 Konsekvent

Denna riktlinje lider lite av samma problem som den som behandlar den konceptuella modellen. Om man poängterar att det är viktigt att system och program är konsekventa med tidigare versioner blir det svårt att nå någon reell förändring. Ett annat problem är att många program inte är konsekventa

med de riktlinjer som finns. Olika kortkommandon betyder olika saker i olika program, och skiljer sig ibland även beroende på språk. Detta är t ex den funktionen som Tognazzini prioriterar som i högst behov av att vara konsekvent.

5.1.7 WYSIWYG

Tanken bakom WYSIWYG är väldigt god. Man skriver ett dokument eller ritar en bild och resultatet när det skrivs ut blir precis som det ser ut på skärmen. Problemet med detta är att man går miste om mycket av fördelen med att använda en dator. En dator kan göra så mycket mer än att skriva text. En länk i ett Internetdokument blir vid utskrift bara ett understruket ord medan det på datorn håller mycket mer information. Samma sak skulle kunna gälla referenslistor eller olika typer av objekt som i sig kan hålla ytterligare information. I och med att man utgår ifrån WYSIWYG riskerar man att gå miste om möjligheter att skapa multifunktionella objekt och komma ifrån tänkandet kring att allt är en representation av ett papper.

5.1.8 Förlåt användaren

Eftersom vi redan talat lite om den problematik under 5.1.4 vill vi här istället föra fram en kanske något radikal ståndpunkt. Datorn har en väldigt dålig förståelse över vad vi som användare vill ha utfört. Ett system som WIMP, som bara tar emot indata från tangentbord och mus är dömt att misslyckas då det aldrig kan känna av vad användaren egentligen är ute efter när denne gör fel. Detta resulterar i felmeddelanden, som ofta inte alls hjälper användaren, på så sätt att datorn inte reder ut problemet åt en.

5.1.9 Uppfattad stabilitet

I riktlinjerna förespråkas stabilitet genom vissa element som är ständigt närvarande. Man kan även koppla detta till det som Tognazzini kallar för *Track state*. Att systemet bör spara information om var användaren befann sig senast, så att när denne återvänder till sitt arbete direkt kan uppta det utan dröjsmål. Istället när vi slår på en dator idag så möts vi av skrivbordet och måste på nytt starta de program vi senast arbetade med.

5.1.10 Lägeslöshet

Det här en riktlinje som låter väldigt bra och som är svår att invända mot, problemet är att den inte följs i praktiken, att utvärdera det är dock inte syftet för vår undersökning.

5.1.11 Sammanfattning

Det finns flera tydliga brister kring användbarhet i WIMP-baserade system. Dessa brister är ofta interrelaterade, olika punkter går in i varandra. Vi menar att de allvarligaste bristerna kan sammanfattas i följande tre punkter. Det är dessa vi sedan vill studera alternativa lösningar till.

- *Problem med metaforer och den konceptuella modellen.* Det allvarligaste problemet här är att man utgår ifrån att alla användare är nybörjare. För att ändå ge erfarna användare en möjlighet att effektivisera sitt arbete har man lagt till särskilda kortkommando och liknande. Vi menar att denna utgångspunkt idag inte är helt korrekt. En så stor andel av befolkningen, är väl bekanta med datom att det måste finnas en sätt att utveckla ett system som bygger på en effektivare grund.

- *Direktmanipulation.* I WIMP använder man en teknik som att objekt på skärmen hela tiden direkt hanteras via input från pekdon eller dylikt. Vissa aktioner av denna typ borde kunna vara automatiserade. Vidare är man beroende av särskilda program för att ändra i vissa typer av objekt. För att ändra i ett dokument måste vi först öppna ett ordbehandlingsprogram, och sedan manuellt genomföra varje ändring.
- *Användarkontroll och återkoppling.* Av riktlinjerna kan man ana en viss motsägelse kring detta område. Man vill att användaren ska utforska systemet så mycket som möjligt, samtidigt som man vill begränsa möjligheterna beroende på vad användaren klarar av. Här krävs mer tydlighet och en uttalad strategi om vad avsikten är. Man borde ge användaren sken av att ha full kontroll samtidigt som det finns tydliga ramar och regler som styr hur denna kan utövas. Dessa regler och ramar måste också tydliggöras för användaren genom kontinuerlig feedback.

6 Post-WIMP?

Vi har nu identifierat och belyst olika problem med WIMP, och därför vill vi diskutera kring ett par system som är konstruerade utifrån andra principer. Dessa använder vi för att exemplifiera och visa på hur man faktiskt kan lösa några av de problem som vi påvisat.

6.1 Introduktion till systemen

Vi inleder med en ingående presentation av varje system för att erbjuda en bakgrund till den diskussion som sedan följer.

6.1.1 Archy

Archy är ett ännu icke färdigutvecklat operativsystem, som erbjuder flera intressanta idéer kring hur man kan styra en dator. Upphovsmannen till systemet är Jeff Raskin. Med Archy är det tänkt att Raskin ska realisera alla sina idéer om hur ett system med hög grad av användbarhet bör fungera. Archys gränssnitt byggs och utvecklas efter kognitionsvetenskapliga idéer för att hjälpa människor att kommunicera med sina datorer. Målet för utvecklarna är att skapa mer effektiva och enklare tillvägagångssätt för att få tillgång till datorns kapacitet (Raskin 2006a). Archy är till för alla tänkbara användare, så som synskadade, äldre, unga och personer med olika datorvana.

Problemet med dagens grafiska operativsystem är enligt Raskin att det kräver att man förstår systemets dåliga vanor, tekniska tillvägagångssätt, hjälpsystem, program från tredje part m m. Utvecklarna bakom Archy menar även att dagens GUI är komplexa sammansättningar av tekniker, som kräver moderna och kraftfulla datorer bara för att köra gränssnittet. De menar dock inte att arbeta direkt i en terminal, d v s med endast textbaserade kommando är enda lösningen. Detta då det ofta är svårt att lära sig och lägger en stor börda på användaren som måste memorera mängder med kommandon. Bortsett från det är textbaserade gränssnitt snabba och använder datorns resurser betydligt mer effektivt. Avsikten med Archy är därmed att skapa ett nytt sorts gränssnitt som är lika effektivt som ett textbaserat, men samtidigt lika enkelt att lära sig som ett grafiskt (Raskin 2006a).

Den idag färdiga delen av Archy är en fungerande textredigerare, byggt på en rad principer för användbarhet. Principerna gäller dock även för det grafiska gränssnittet, vilket dock endast finns tillgängligt som en prototyp (Raskin 2006c). När Archy blir färdigutvecklat är det tänkt att ersätta ett helt operativsystem. För att navigera mellan dokument och andra objekt, inför Archy ett helt nytt paradigm, nämligen Zooming Interface Paradigm [ZIP]. Istället för det traditionella sättet att navigera bland filer och dokument, ska man zooma in innehållet och direkt kunna manipulera det på plats, utan att byta mellan applikationer. Raskin (2002) beskriver dagens operativsystem som en labyrint, där varje dörr är betecknad med ett kort och ofta kryptiskt namn. Självklart kan man inte se vad som finns bakom nästa dörr, förrän du öppnat dörren och klivit in i rummet. När du väl klivit igenom dörren, kanske du inte ens kan se vilket rum du kom ifrån. Den här jämförelsen gör Raskin både när han talar om WIMP-baserade gränssnitt i en dator och navigering på webbsidor. Raskin beskriver vidare ZoomWorld som ett bättre sätt att navigera på. Med ZoomWorld gör man det möjligt att se sitt mål redan när man bestämt sig för vilket det är. ZoomWorld löser problemet med begränsad arbetsyta på skärmen, genom att man zoomar in det man vill manipulera. Allt man kan komma i kontakt med finns

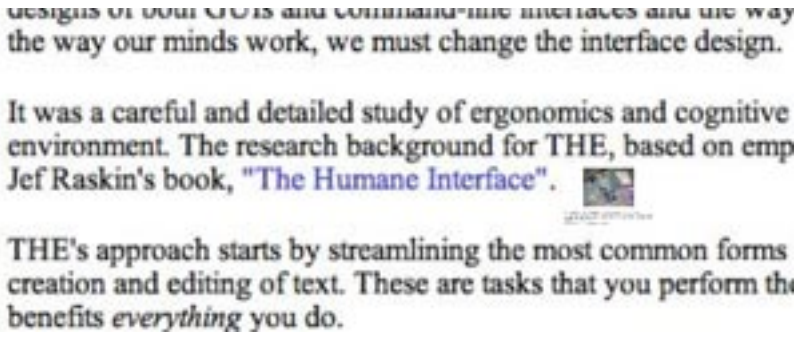
någonstans i ZoomWorld. Man kan tänka sig en labyrint, så som vi beskrev navigeringen i ett vanligt operativsystem, där man kan flyga ovan labyrinten och direkt se vilka vägar som är möjliga att gå, samt vad som befinner sig bakom varje dörr. Då slipper man problemet med stängda dörrar och kryptiska beteckningar och kan direkt gå till vårt mål. På detta vis förklarar Raskin syftet med ZIP.

Figur 6.1: Exempel på arbetsyta i ZoomWorld (Raskin 2006c)

Man kan tänka sig Figur 6.1 som sin arbetsyta, d v s det som idag ses som ett skrivbord på skärmen. Texten och bilderna ses som innehåll och kan placeras var som helst på skärmen. Raskin har i sin demonstration valt att gruppera bilder för sig och text för sig. Arbetsytan är ej begränsad, då man kan zooma ut och få tillgång till ytterliggare arbetsyta när och var som helst. Texten och bilderna i figuren blir då givetvis mindre. Vid inzoomning av texten ”THE Related” (Figur 6.2) visas vad som nu finns på skärmen som arbetsyta.

Figur 6.2: Nytt utgångsläge i ZoomWorld. En bild uppenbaras i texten under ”THE Is Not An Editor So What Is It?" (Raskin 2006c)

Inzoomningen i Figur 6.2 är så nära att man kan se ytterliggare nytt innehåll: ”THE Is Not An Editor… So What Is It?”, ”Screenshot”, ”Joining The THE Team”. Om man fokuserar på det första innehållet: ”THE Is Not An Editor… So What Is It?” och zoomar in på det, uppenbaras en bild i Figur 6.3.



Figur 6.3: Nytt utgångsläge i ZoomWorld.
En bild uppenbaras i texten under "THE Is Not An Edito So What Is It?"

Med ZoomWorld ska det vara enkelt att redigera det innehåll man fokuserar på, men även lika enkelt att skapa nytt innehåll. Det framgår inte helt klart hur man ska gå till väga för att lägga till innehåll i ZoomWorld, men man kan enkelt antaga att man övergår till t ex textredigeraren när man vill skapa ett textdokument, och bildredigeraren för bilder. Innehållet placeras då där man zoomat in och man kan självklart välja att flytta och organisera efterhand.

Figur 6.4 visar hur man kan zooma in på bilder som kan vara placerade i en text och vara större än en bokstavs storlek. Att gå ett steg längre skulle vara att zooma in bildtextentill läsbart fokus.

Figur 6.4: Bild inzoomad
(Raskin 2006c)

ZoomWorld bygger på teorin att människor har lättare för att minnas platser med hjälp av landmärken. Inte helt främmande varandra är ZoomWorld och ett planeringsrum. Konceptet är detsamma, då man fyllt väggarna i rummet med kom-ihåg-lappar, foton, tidningsblad eller vad som helst, som hjälper oss att komma ihåg bättre. Raskin beskriver det som när man kommer in i ett rum, står man i mitten och ser sig omkring, kanske fastnar för ett speciellt foto och kommer ihåg att det var där man placerade det viktiga tidningsklippet som man letat efter. Raskin menar att anledningen till att vi hittar saker så lätt i ett sådant rum, beror just på att vi med hjälp av landmärken lättare kan orientera oss och komma ihåg relativa platser. Figur 6.1 visar hur man kan organisera sitt innehåll i ZoomWorld, med text på ena sidan och bilder på andra sidan.

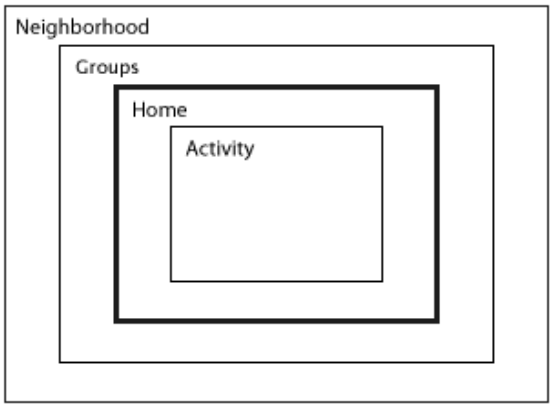
6.1.2 OLPC

One Laptop Per Child [OLPC] startade det intressanta projektet att tillverka en dator, till en maximal

kostnad av $100. Datorn som kallas XO, riktar sig främst till barn i utvecklingsländer, som ofta inte har råd att köpa en dator. Grundaren, Nicholas Negroponte (OLPC, 2007a), skriver att det är ett projekt inriktat på utbildning. Det är alltså inte själva datorn som står i fokus för OLPC, utan snarare att kunna utbilda fattiga barn med hjälp av datorn. Enligt dem själva, ger datorn nya möjligheter för barnen att upptäcka, experimentera och uttrycka sig själva.

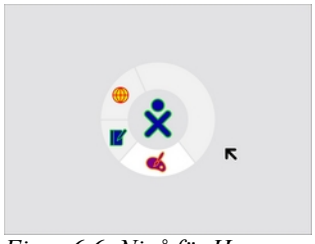
Datorn blir intressant för oss, tack vare det nya gränssnitt som utvecklats för att fylla detta specifika behov, med barns lärande i fokus. OLPC beskriver gränssnittet som revolutionerande, på så sätt att det har nätverket i fokus. I fokus ligger nämligen nätverket, med kommunikations- och utbildningsverktyg som redskap för att utbilda barn. Operativsystemet heter Sugar och har tagits fram tillsammans med utvecklare från Pentagram och Red Hat. Systemet bygger designmässigt på en rad principer vilka kan liknas vid Apples riktlinjer som vi tidigare .

Gränssnittet Sugar UI bygger på en helt ny typ av metafor. OLPC kallar själva metaforen för ”Zoom Metaphor”. I Figur 6.5 ser vi att metaforen består av *Home*, *Group*, *Neighborhood* och *Activities*.

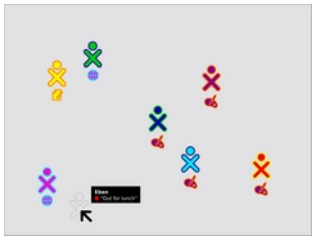


Figur 6.5: Zoom-metaforen i Sugar UI (OLPC 2007a)

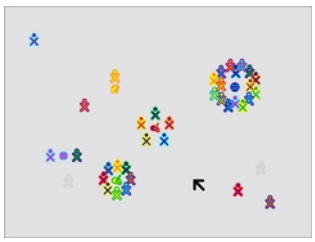
OLPC kallar detta för olika zoom-nivåer (OLPC a, 2007).



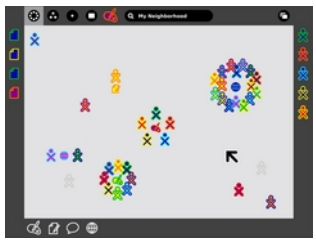
Figur 6.6: Nivå för Home



Figur 6.7: Nivå för Group



Figur 6.8: Nivå för Neighborhood



Figur 6.9: The Frame är nu tillgänglig

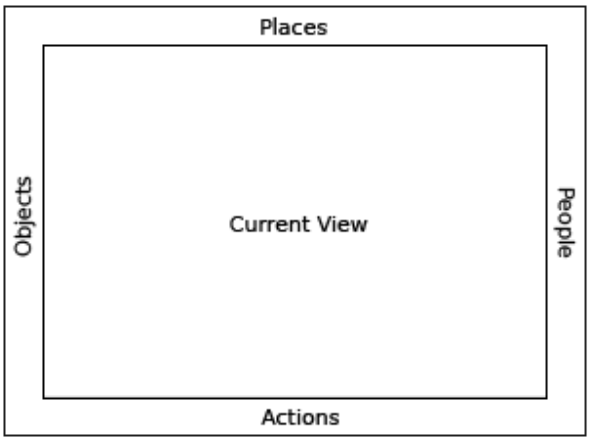
Den nivå som kallas för *Home,* Figur 6.6 är den nivå som mest kan liknas med den traditionella skrivbordsmetaforen. Den här nivån är den första som visas när man startar datorn och innehåller de aktiviteter användaren har tillgång till. Det är en minimalistisk miljö man möts av. De val man har att välja mellan framgår tydligt utan att man behöver leta. Användaren kan välja mellan att zooma ut till antingen Gruppnivån eller Grannskapet för att hitta vänner och andra aktiviteter än sina egna, eller zooma in på sina egna lokala aktiviteter som befinner sig under *Home.*

Group-nivån, Figur 6.7, är ett steg tillbaka från Hemnivån. Här får användaren en möjlighet att kontakta sina vänner, klasskompisar eller andra grupper. Det finns här också möjligheter till att

redigera sina grupper, genom att ta bort, lägga till och modifiera andra användare.

Ännu längre bak finns *Neighborhood,* Figur 6.8, och här har användaren tillgång till hela det lokala nätverket och han eller hon kan se precis vilka som är tillgängliga. Man kan även se inom vilka aktiviteter som en användare är tillgänglig. I den här nivån kan man söka upp både användare och aktiviteter. Aktiviteter kan vara t ex chattar eller datorspel.

The Frame, Figur 6.9, blir tydlig genom ett kortkommando på datorn. Exempelvis om du drar pekaren ut mot kanten, träder ramen fram och blir tillgänglig. Denna ram består av fyra olika kategorier: *Places*, *People*, *Activities* och *Objects* vilket visas i Figur 6.10. *Places* innehåller alla de nivåer som finns i och representerar Zoommetaforen (Home, Grourps, Neighborhood och Activities). Till vänster i ramen finner vi Objects, vilket innebär sådant som användaren utforskat nyligen. Det kan vara texter, spel, diskussioner eller andra aktiviteter. För att starta en aktivitet kan man utgå från ramens nederkant, som är till för att starta nya aktiviteter. Till höger i ramen har användaren tillgång till en lista över alla sina vänner.



Figur 6.10: The frame och dess organisation (OLPC 2007a)

6.1.3 Type Managers

En Type Manager kan ses som ett mindre radikalt alternativ till system som kräver att användaren jobbar med filer. Mayer (2005) förklarar Type Managers som i likhet med det vi kallar filhanterare. Likheten ligger i det syfte som både filhanterare och Type Managers har, nämligen att låta användaren utforska filer på hårddisken. Men istället för att hantera alla filer manuellt i systemet, tillåter en Type Manager att man endast hanterar en viss typ av filer. Man kan se en Type Manager som ett program utvecklat med funktioner anpassade för en viss typ av filer. Det kan vara bilder, e-postmeddelanden, textfiler o s v. Denna typ av program är ingen nyhet. Det är däremot trenden att programmen exklusivt hanterar en viss typ av filer. Inte bara för att redigera eller skapa, utan även för att organisera, något som traditionellt utförts med filhanteraren i operativsystemet. Exempel på denna

Figur 6.11: iPhoto är en Type Manager för bilder (Apple 2007b)

typ är bl a Microsoft Outlook, iTunes och iPhoto. I Figur 6.11 ser vi iPhoto som är en typisk Type Manager.
Det som definierar en type manager enligt Meyer (2005) är bland annat följande punkter.

- En Type Manager ska hålla användaren borta från detaljer som rör filsystemet. Det ska finnas funktioner som automatiskt sorterar filerna på hårddisken.
- Type Managers ska tillhandahålla funktioner för att redigera och visa metadata för filer.
- De ska innehålla verktyg för att importera och exportera data. Exempelvis ska ett CD-bränningsverktyg kunna bränna CD-skivor och bildhanterare ska kunna importera bilder från en digitalkamera.
- De ska även innehålla enklare funktioner för manipulering av data.
- Type Managers ska innehålla full sökbarhet av filer och metadata.
- Det ska finnas möjlighet att gruppera data. Exempel kan vara att bildhanterare ska kunna visa de senast importerade bilderna.
- Type Managers ska visa upp väl definierade och domänspecifika gränssnitt för andra program, snarare än att implementera dem själv.
- Type Managers ska vara enkla att bygga ut och tillhandahålla ramverk för utbyggnad av andra verktyg och program.

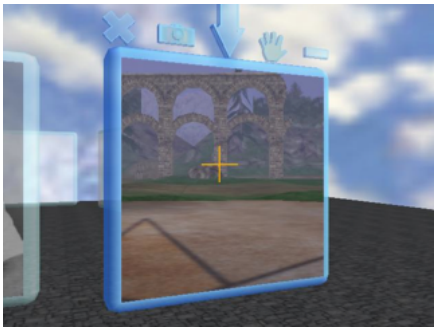
6.1.4 Croquet

The Croquet Consortium är en ideell förening ämnad åt att utveckla och gynna öppen källkod, Croquet-teknologier för forskning, utbildning och industrier (The Croquet Consortium, 2007a). Målet för Croquet-projektet är att utveckla ett globalt nätverk med intresse bland utövare och institutioner för att underlätta och uppmuntra gemensam försörjning. Man vill även tillhandahålla utbildning och forskning gällande användandet av Croquet och vidare vill man starta projekt baserade på Croquet för att sedan distribuera dessa.

Croquet är ett operativsystem i 3D-miljö som används för programutveckling eller andra kollaborativa aktiviteter. Med Croquet ingår man i ett nätverk med flera Croquet-användare, där man delar resurser med andra medlemmar i nätverket. Croquet främjar möjligheten att låta flera användare verka i samma miljö då man synkroniserar sin egen dator med andras, och skapar 3D-miljöer som andra kan ta del av. Smith et al. (2003 s. 3) skriver att Croquet kan liknas med hur informationen på Internet hanteras på många sätt, där varje användare kan skapa sin egen webbsida för att presentera information. Skapar man ett rum på sin egen dator som använder Croquet, kan övriga medlemmar ta del av detta rum. I Figur 6.12 ser vi ett exempel på detta där två användare inspekterar samma objekt.

Figur 6.12: En kollaborativ miljö med flera användare.
(The Croquet Consortium 2007c)

Precis som för webbsidor på Internet kan man skapa länkar, som leder till andra världar och miljöer. En användare tilldelas rättigheter till andra världar och kan då stiga in i denna (Smith et al. 2003). I Figur 6.13 ser man en portal som ett fönster innehållande en annan värld. Man kan alltså redan innan man klivit in i den nya världen se vad som döljer sig bakom. Världarna kan användas till mycket, bland annat virtuella rum där man utför olika aktiviteter, konferensrum, landskap eller vad som helst som går att illustrera med 3D. Vidare beskriver Smith et al. en portal som en tredimensionell koppling mellan olika världar. Den stora skillnaden mellan rum och världar i Croquet och rum i verkligheten är att rummen i Croquet inte har någon geografisk belägenhet. Spegeln i Figur 6.12 är också en portal, men skillnaden är att denna portal är kopplad till sig själv, vilket medför att man ser sig själv när man står framför den.



Figur 6.13: Portal till en annan värld (Smith et al. 2003)

Det som skiljer Croquet från liknande program, så som Second Life, är att Croquet är ett verktyg för att utveckla annan programvara och är därmed mycket mer flexibelt. I Second Life skapar man inte en datormiljö som tillhör användaren själv, utan använder en mer begränsad miljö med begränsad tillgång till val av miljöer. Vidare har Croquet inte en speciell värddator för att tillhandahålla teknologin, så som 3D-miljön. Däremot tillhandahåller Croquet programmeringsspråk och har en egen integrerad utvecklingsmiljö, vilket gör det bra för mjukvaruutveckling.

Enligt utvecklarna (The Croquet Consortium, 2007b) är Croquet intressant för att det blev en så pass lyckad infallsvinkel för att styra en dator. Det kan med andra ord fungera som vilket operativsystem som helst. De anser att det traditionella sättet att styra en dator på med WIMP, är utdaterat och den enda skillnaden man kan se på en PC från 2004 och en av dem första Macintosh-datorerna är färgerna. Man går också ett steg längre och ger ”Croquet” som svar på frågan: ”Om vi var i tillstånd till att utveckla ett nytt operativsystem och gränssnitt, med den kunskap vi besitter idag, hur långt skulle vi kunna gå?”.

6.2 Alternativens fördelar

Vi kommer nu nedan att utifrån ett användbarhetsperpektiv visa på några av de ovan beskrivna systemens fördelar.

6.2.1 Metaforer

Vi har redan konstaterat i kapitel 5.1.1 att metaforer kan medföra en minskad effektivitet vid användandet av system och vi vill nu ge exempel på hur detta kan gå till. Filer och mappar i datorn är metaforer för just filer och mappar i verkligheten. Dessa metaforer följer användarens konceptuella

modell, så som vi beskrev i kapitel 5.1.2, men är detta bra eller dåligt? Användarens konceptuella modell är idag starkt påverkad av system som utvecklades för många år sedan. Vi tror att filer och mappar lever kvar för att användaren inte känner till något annat sätt att hantera innehåll. Liksom Raskin (2000 s. 118) påpekar, är vi begränsade när vi namnger våra filer. Det tenderar till att bli korta och kryptiska beteckningar, eftersom vi ofta är stressade och alltid begränsade till hur långa våra filnamn kan vara.

Att spara en fil innebär nästan alltid ett avbrott från vårt egentliga arbete (Raskin 2000, s. 118). Sparar vi en fil för första gången, innebär det ett avbrott vilket resulterar i en rad dialogrutor och fokus ägnas nu åt att komma på ett bra filnamn. Bra i den bemärkelsen att när vi nästa gång ska öppna samma dokument, behöver vi komma ihåg filen med den beteckning som beskriver dokumentets innehåll. Raskin (2000) för ett liknande resonemang när han beskriver avbrottet och filnamnet som det största problemet. Även detta är emot Nielsens (1993b) princip om att *minimera användarens minnesbelastning*.

Ett annat problem med filer, är att den mentala bild vi tar med oss till datoranvändandet, inte stämmer överens med hur metaforen fungerar i WIMP. I verkligheten arbetar vi med en fil, låt oss säga dokument, men i datorn finns det egentligen två filer för samma dokument. Den ena filen ligger på hårddisken och den andra filen ligger i minnet som en kopia. Cooper (1995, sid. 84) menar att den mentala bild vi har på en fil, faller på två sätt när vi jobbar med filer i datorn. Det ena är att det alltid finns två kopior av filen och den andra är att de båda tillhör programmet. Problemet är då, enligt Cooper att användaren tror att han eller hon jobbar med en fil, medan man i själva verket jobbar med två filer. När användaren gör ändringar i filen, görs ändringarna i kopian, vilken ligger i datorns arbetsminne.

Ett alternativt sätt att arbeta med dokument och samtidigt kringgå användandet av metaforer som filer och mappar är att endast fokusera på vad det är som dokumentet innehåller. ZoomWorld i Archy löser problemet med filer genom att låta operativsystemet hantera de åt en. När man utvecklat Archy, har man hela tiden haft som princip att användaren direkt ska arbeta med innehållet och hela tiden hålla fokus på det. Man har helt uteslutit metaforer för filer och fokuserar istället på ett dess innehåll som vi kan se i Figur 6.1 där innehållet är presenterat direkt på skärmen.

Vid sökning av text i Archy använder man sig utav en sökfunktion för att hitta rätt dokument. Det finns med andra ord ingen filhanterare där man navigerar mellan olika filer och mappar. För att visuellt leta efter dokument använder sig Archy utav ZoomWorld. Detta innebär att man varken jobbar med filer, mappar eller program. Således behöver man aldrig tänka på att komma på unika och förutsägbara filnamn. Vi tror att detta kan vara ett alternativ som hanterar användarens minnesbelastning på ett bättre sätt och därmed kan ses som ett effektivare användande. Detta ser Nielsen (1993b) som två mycket viktiga och fundamentala principer.

Tognazzinni (2003) menar att det är viktigt att gränssnitt är lätta att utforska (*Explorable Interfaces*). Man bör alltså kunna navigera genom gränssnittet med hjälp av t ex. landmärken för att lättare kunna hitta rätt. Vidare ska användaren enkelt kunna återgå till en ursprungsplats när han eller hon vill. I dagens filhanterare finns det visserligen ofta genvägar tillbaka till ursprungsläge, men detta krävs att man först hittar dessa genvägar och sedan klickar på dem. Det är också svårt att hitta unika landmärken som hjälper en med orienteringen i en typisk filhanterare.

Med hjälp av ZIP (t ex. ZoomWorld) kan användaren hela tiden ha illusionen att han eller hon alltid är på rätt ställe och informationen förs till denne. Detta är en annan viktig princip som Tognazzini för fram och som han kallar för *Visible navigation*. Informationen finns där och användaren behöver bara fokusera på rätt ställe. Motsatsen till detta är när man surfar på Internet med en webbrowser, och man har ingen aning om var man är. Likså med Findern i Mac OS X (10.4.x), som praktiskt taget saknar

överblick om var man befinner sig i filsystemet. Med ZoomWorld är tanken att man alltid ska veta var man befinner sig, tack vare att man lärt sig memorera platser genom landmärken.

Ett annat problem med metaforer är att om man inte känner till den verklighet metaforen avbildar, hjälper den oss inte att förstå systemet. OLPC (2007b) menar att det är viktigt att känna till vilka som kommer att använda systemet och har därför tagit fram ett nytt sätt att se på metaforer eftersom deras användare har en mycket begränsad kunskap från skrivbordsmiljöer.

OLPC menar att man måste ha kunskap om vem man utvecklar för. Målet för OLPC är att man vill skapa möjligheter för användarna, barnen i detta fall, att utforska, experimentera och uttrycka sig själva. OLPC känner väl till sin målgrupp och menar att många av de här barnen, har haft väldigt liten, eller ingen kontakt med datorer tidigare. De har alltså ingen aning om hur man interagerar med en persondator. Detta medför således att de även har liten erfarenhet av de traditionella metaforer som används i västvärldens olika system. OLPC skriver därför att utvecklarna måste p ena sidan fokusera på att skapa gränssnitt som tillåts att utforskas, och å andra sidan utveckla nya metaforer som beskriver gränssnittet för denna specifika målgrupp.

Vidare menar utvecklarna bakom OLPC att skrivbordsmetaforen passar bäst i kontorsmiljö, eller i ett västerländskt hem och har därför blivit tvungna att adoptera en ny sorts metafor, vilken i sig framhäver *communityn*. Samtidigt som man kan se klara likheter mellan Sugar UI och dagens skrivbordsmetaforer, finns det klara skillnader. Följande tabell visar skillnader hur man ser på olika kontexter så som skrivbordsmetaforen, menyer, filsystem, program och filer. Man kan se det som alternativa synsätt.

Tabell 6.1: Jämförelse mellan synen på metaforer i WIMP och OLPC XO

WIMP	**OLPC XO**
Skrivbordsmetafor	Neighborhood
Menyer	The Frame
Hierarkiskt filsystem	Journal
Program	Activities
Filer	Objects

Sugar UI bygger som bekant på en zoom-metafor, vilken i sig kan ses som en mer global och abstrakt metafor. Inte lika stark förankrad i verkligheten som skrivbordsmetaforen. Med global menas att OLPC:s begrepp så som Neighborhood och Activities kan ha lika stor betydelse världen över. Övriga begrepp i Tabell 6.1 så som The Frame, Journal och Objects går att lära sig utan att de kopplas till verkligheten.

Vi tror också att vi i västvärlden har en så stor kunskap om hur datorer och operativsystem fungerar, att metaforer som skrivbordsmetaforen ger fler problem än fördelar. Datorn XO kanske inte är det alternativ vi efterlyser, men tänkandet om en global metafor, likt den som används i Sugar UI skulle kunna bekämpa många problem som uppstår i och med skrivbordsmetaforen. Särskilt avseende filer och mappar.

Ett annat nytt och intressant sätt att se på metaforer är använda virtuella 3D-miljöer. Croquet är ett

operativsystem som bygger på 3D-miljöer. Man kan beskriva Croquet genom att ta skrivbordsmetaforen till en högre nivå och implementera en hel värld. Som metafor är man då inte bunden till att endast använda t ex filer och mappar. Varför inte bygga en vägg där man kan samla familjens fotografier. Eller ett konferensrum, inrett med arbetsmaterial på väggar och bord, och som man delar med sina arbetskollegor. En värld där användaren skapar sin egen omgivning utifrån sina behov och inte är bunden till någon speciell metafor. Om man jobbar med många textdokument, kanske man vill ha ett virtuellt rum för endast detta ändamål, och ett annat rum för bilder. Den stora fördelen med Croquet ligger just i samverkan mellan användare, navigering och att användaren fritt kan utforska systemet utan att hindras av felmeddelanden och dylikt. Som vi kunde se i vår teorigenomgång är ett utforskbart gränssnitt något att efterfråga, och mer utforskande än Croquet är svårt att tänka sig. Inte nog med att navigeringen på den lokala datorn uppmanar till att utforska och memorera landmärken, det fungerar likadant när man utforskar andra användares datorer. Den lokala datorn och datorer på olika nätverk är synkroniserade, på så sätt att om användaren väljer att göra entré hos en annan användare/dator upplever de samma miljö. Detta leder inte till ett konsekvent gränssnitt, vilket är en viktig riktlinje i WIMP, men det resulterar i ett gränssnitt som är lätt att utforska och där man kan ha bra samverkan med andra användare. I WIMP och skrivbordsmetaforen ser man användare på andra nätverk som objekt, vilka man kan manipulera och utforska om man vill. Med Croquet blir man en del av den virtuella världen och gränserna för vad som ligger lokalt på datorn och på nätverket suddas ut.

6.2.2 Direktmanipulation

Men för att komma från problemen med direkmanipulation på objekt (kapitel 5.1.3) behöver man ta till nya sätt att manipulera data med. Den stora fördelen med tänkandet hos Archy är att fokus alltid ska ligga på innehållet, det som ska manipuleras. Ofta är det direktmanipulation som gör att fokus försvinner från det egentliga arbetet, då det ofta krävs att man flyttar handen från tangentbordet till pekdonet för att markera ett annat objekt i skrivbordsmetaforen, t ex öppna tredjepartsapplikationer, miniräknare eller navigera efter en fil på hårddisken o s v. Detta tar givetvis tid och för bort fokus från det egentliga innehållet man tänkt att arbeta med. ZoomWorld undviker direktmanipulation. Med ZoomWorld kan användaren utforska hela datorns innehåll utan att direkt manipulera objekt, så som man behöver med navigering med *Explorer* i Windows, eller *Findern* i Mac OS X, som är två filhanterare hos respektive operativsystem.

Raskin (2006b) radar upp en lista med problem som kan förhindra eller störa användaren i sitt egentliga arbete i dagens operativsystem. Här är några av punkterna från projektets hemsida (Raskin 2006b):

> Du skriver ett mail till en vän om din semester i Tahiti, när du upptäcker att bilden du vill bifoga är defekt. Helst av allt skulle du vilja fixa felet på plats, utan att navigera efter filen och öppna den i ett nytt program, men det är givetvis inte möjligt med dagens WIMP-baserade gränssnitt.
>
> Dagens operativsystem tillhandahåller inga sådana funktioner, utan att det krävs tredjepartsapplikationer. Man behöver också leta rätt på bilden för att kunna manipulera den, s.k. *direktmanipulation*.
>
> Du behöver göra en uträkning samtidigt som du skriver på ett dokument, men du vill för den sakens skull inte behöva starta miniräknaren på datorn.
>
> Precis som i föregående punkt, måste man flytta fokus från dokumentet. Och i värsta fall leta efter miniräknaren i en katalog eller meny.

Du jobbar med något viktigt på datorn, när helt plötsligt du råkar ut för de klassiska dialogrutorna. För att gå vidare måste du antingen välja Avbryt eller OK.

Återkoppling och kommunikation är en viktig princip i WIMP, men vi tror att det, i dagens operativsystem inte går att ge bra information via dialogrutor och fortfarande ha ett *effektivt användande*, som är en av Nielsens riktlinjer. *Bra och effektiva felmeddelanden* är också viktigt, enligt Nielsen (1998). I Microsoft Windows XP, är det inte ovanligt med att man får upp en dialogruta där man endast kan välja ”OK”. Eller en dialogruta som inledes med en lång text och sedan efterföljs med två val: ”YES” och ”NO”. För att svara ja eller nej här, krävs det givetvis att användaren måste läsa igenom texten och förstå den. En bättre lösning enligt vårt tycke, skulle vara att ha alternativen ”Don't Save” och ”Save document”, om det handlar om att spara dokument. Då behöver användaren inte fokusera på dialogrutan lika länge, utan ser då de båda alternativens innebörd väldigt snabbt.

Du vill utföra en uppgift, men du tvekar hur du vill utföra den eftersom det finns flera tillvägagångssätt som utför samma sak.

Norman (1988) skriver i en av sina sju punkter (*Exploit the power of constraints both natural and artificial*) att användaren bara bör känna till en möjlig väg för att utföra en uppgift. Detta just för att inte förvirra användaren.

Eftersom Archy är utvecklat utifrån att lösa bland annat dessa problem, har man helt försökt att gå ifrån tänkandet med direktmanipulation. Vi anser att det många gånger är onödigt att manipulera objekt, i de fall då det är innehållet man vill åt.

En annan viktig princip i Archy är att man ska inte ska behöva lära sig en mängd kommandon för att utföra de enkla arbetsuppgiftema. I vanliga operativsystem måste vi först lära oss mängder med kommandon för att hantera grunderna i det. Ofta använder sig program utav olika uppsättningar av snabbkommandon, menysystem o s v. Resultatet av detta är ofta frustration och felmeddelanden. Archy eliminerar detta problem genom att inte låta operativsystemet vara synligt för användaren. Istället låter Archy användaren utföra sina kommandon och operationer direkt på plats, vilket betyder att man inte behöver fokusera på ett nytt program med nya tillstånd och kommandon. Fokus kvarstår på den egentliga arbetsuppgiften. För att möjliggöra detta, använder Archy ett relativt litet set med fundamentala operationer som tillåter användaren att utföra en mängd uppgifter inom ett stort område.

Archy kan upplevas som en ganska radikal övergång, från filsystem. Ett mindre radikalt alternativ, men med samma grundidé när det gäller att dölja hanteringen av filer för användaren, är så kallade Type Managers. Visserligen är dagens Type Managers utvecklade för att interageras med hjälpa av WIMP. Men dessa tycker vi ändå kan ses som en intressant utveckling. Direktmanipulation är tidskrävande, speciellt när vi har flera hundra- eller tusentals filer att hantera.

6.2.3 Användarkontroll och återkoppling

När det gäller användarkontroll tror vi att det behövs, men att man kan utnyttja systemen bättre genom att låta dem ta beslut som användaren skulle behöva en längre tid på sig att fatta. En samverkan mellan användarkontroll och där systemen fattar allt större beslut tror vi är nödvändigt. Vi vill också påstå och säga att ZoomWorld leder användaren till sitt mål genom att ge användaren ett sken av att ha full kontroll, medan det är begränsat till den grad att man inte kan hantera och flytta runt filer. Här tar Archy ett större ansvar och sköter detta åt dig. När Tognazzini (2003) skriver om *Exploreable Interfaces*, menar han just att användaren ska tro att han eller hon har full kontroll, men att systemet hela tiden erbjuder den närmaste vägen till målet och i ZoomWorlds fall blir det genom landmärken.

7 Sammanfattande diskussion

Vi har i vårt uppsats genomfört en explorativ studie av alternativ till WIMP-baserade gränssnitt. Vi har funnit ett fåtal system som vi bedömt kan erbjuda intressanta nytänkande idéer kring gränssnitt för operativsystem. Dessa har vi redogjort för grundligt för att skapa förståelse för ett ämne som vi tror att många saknar insikt i. Vi har vidare för att visa på behovet av sådana utvärderat WIMP ur ett användbarhetsperspektiv samt försökt ur samma perspektiv diskutera alternativens fördelar. Vår bedömning och utvärdering menar vi bör endast ses som en inledning till en diskussion, vi har inte haft någon ansats att dra långtgående slutsatser eller att presentera ett oomtvistligt resultat kring dessa system som har en sådan enorm spridning. Vi hoppas dock att vi kunnat bidra till nya tankebanor kring användandet av våra datorer och kan se flera uppslag till vidare forskning. Det skulle t ex vara intressant att komplettera vår undersökning med olika typer av empirsika tester. Den utvärdering vi genomfört har baserats kring metoden heuristisk utvärdering, en metod där man försöker uppskatta hur användare ska reagera och vilka problem som kan tänkas uppstå. Särskilt intressant skulle det naturligtvis vara att då genomföra en undersökning där man försöker klargöra hur användare faktiskt uppfattar de olika systemen, t ex genom olika observationer eller enkäter.

Angående våra resultat finner vi det anmärkningsvärt att vi endast genom en inspektion utifrån välkänd teori kan visa på att det finns tydliga brister i användbarheten hos de operativsystem en så stor del av världen använder dagligen. Vi har vidare diskuterat och visat klara brister med skrivbordsmetaforen samt påvisat att det finns alternativ som erbjuder annorlunda lösningar. Vårt undersökningssätt, eller vår metod om man så vill, har dock visat sig vara för begränsad för att kunna göra en konkret bedömning av vilka system som har högst grad av användbarhet. Naturligtvis också beroende på att vissa alternativ inte är färdigutvecklade och en jämförelse därför har varit svårgenomförd. Vi tycker dock att det finns indikationer på att alternativen med sina nya grepp erbjuder funktioner som i rätt kontext med säkerhet är mer effektiva. Vi ser även att dagens dominerande gränssnitt börjar anamma vissa nya idéer, man ser t ex fler och fler s k *type managers* och det finns andra liknande exempel som tyder på att de traditionella WIMP-systemen är i förändring.

Utifrån detta menar vi även att man inte kan anta att det är de bästa produkterna som är de mest dominerande, det måste finnas andra bakomliggande orsaker. Dessa har aldrig varit vår avsikt undersöka. Vi har dock i vårt arbete fört fram en tanke om att det delvis kan vara bereoende på användares konceptuella modell. För en användare är datorn idag mångt och mycket synonymt med ett WIMP-baserat gränssnitt, man är väl inarbetad med det, och har en hög förståelse för dess funktioner. Om man då drar användbarhetsdiskussionen vidare kan man på sätt o vis därmed säga att dessa system trots allt har en ganska hög grad av användbarhet, om man jämför med t ex Nielsen definition, där lärbarhet är ett väldigt centralt begrepp. Detta ser vi snarare som ett problem med begreppet användbarhet än som en fördel hos WIMP-systemen. Vissa beståndsdelar är i sig motsägelsefulla. Att produkter ska vara konsekventa med tidigare versioner och lätta att lära, d v s utgå från tidigare kunskap, innebär också att man går miste om många nya tankar. Vi menar därför att endast lärbarheten och vanan vid att hantera en viss produkt inte är tillräcklig för att tala om hög grad av användbarhet. Det går fort att komma igång men om man vill nå en ny nivå tror vi att det är nödvändigt med drastiska förändringar. Dagens system är byggda kring att användare har en konceptuell modell som inte inbegriper datorer, man försöker därför relatera till föremål i världen för att få oss att bättre förstå datorn. Vi menar att datorn idag är en del av den konceptuella modellen. Datorer sätter standarder för hur andra produkter fungerar. Skrivbordsmetaforen bygger kring en tanke att alla datoranvändare är nybörjare. Allt eftersom de lär sig mer finns det avancerade

funktioner och kortkommandon för att effektivisera arbetet. Vi menar att man kanske borde se det från andra hållet istället. Systemen bör vara grundade i en annan föreställning om användares konceptuella modell och därmed vara mer inriktade på effektivitet. Detta system kan då istället erbjuda särskilt lätta funktioner för de som inte förstår systemet i grunden. Det är helt enkelt nödvändigt att tänka i nya banor för att föra utvecklingen framåt. Någon måste ta första steget och skapa den där riktigt bra produkten, som sedan alla tar efter. Eller för att återknyta till titeln, skapa bilen med fem hjul och en propeller.

Referenslista

Tryckta källor

- Carrol, J.M, Mack, R.L & Kellogg, W.A , Helander, M. (ed) (1988). *Handbook of Human Computer-Interaction.* Amsterdam: Elsevier Science Publishers B.V.
- Cooper, A. (1995). *About Face: The Essentials of User Interface Design.* Foster City (CA): IDG Books Worldwide, Inc.
- Faulkner, X (2000). *Usability Engineering*. Basingstoke: Palgrave.
- German, D.M. (2004). *The GNOME Project: a Case Study of Open Source, Global Software Development*. Software Process: Improvement and Practice. Vol. no. 4. s. 201-215.
- Grønmo, S. (2004). *Metoder i samhällsvetenskap.* Malmö: Liber AB.
- Gulliksen, J. & Göransson, B. (2002). *Användarcentrerad Systemdesign.* Lund: Studentlitteratur.
- Halvorsen, K. (1989). *Samhällsvetenskaplig metod.* Lund: Studentlitteratur.
- Holzinger, A. (2005). *Usability Engineering Methods for Software Developers.* Communications of the ACM, vol. 48, no 1.
- Nielsen, J. (1993b). *Usability Engineering.* Chestnut Hill (MA): Academic Press, Inc.
- Norman, D. (1988). *The Design of Everyday Things.* New York (NY): Basic Books.
- Ottersten, I. & Berndtsson, J. (2002): *Användbarhet i praktiken.* Lund: Studentlitteratur.
- Preece, J., Rogers, Y., Sharp, H, Benyon D., Holland, S. & Carey, T. (1994): *Human-Computer Interaction.* Reading (MA): Addison-Wesley.
- Preece, J., Rogers, Y. & Sharp, H. (2002): *Interatction Design.* New York (NY): John Wiley & Sons, Inc.
- Raskin, J. (2000). *The Humane Interface.* Massachusetts: Addison Wesley.
- Smith, D.A., Kay, A., Raab, A., Reed, D.P. (2003): *Croquet – a collaboration system architecture.* Creating, Connecting and Collaborating Through Computing, s. 2-9.
- van Dam, A. *(1997). Post-WIMP User Interfaces.* Communication of the ACM, Vol 40, no. 2, s. 63-67.

Elektroniska källor

- Apple Inc. (2006). *Apples Human Interface Guidelines.* [WWW document] URL http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/OSXHIGuidelines.pdf den9 Maj 2007.
- Apple Inc. (2007a). *Apple – Mac OS X – Automator* [WWW document] URL http://www.apple.com/macosx/features/automator/ den 28 Maj 2007.

- Apple Inc. (2007b). *Apple – iLife – iPhoto* [WWW document] URL http://www.apple.com/ilife/iphoto/ den 28 Maj 2007.
- Croquet Consortium (2007a). *Main Page – Croquet Consortium* [WWW document] URL http://www.opencroquet.org/index.php/Main_Page den 30 Maj 2007.
- Croquet Consortium (2007b). FAQs – Croquet Consortium [WWW document] URL http://www.opencroquet.org/index.php/FAQs 30 Maj 2007.
- Croquet Consortium (2007c). Screenshots – Croquet Consortium [WWW document] URL http://www.opencroquet.org/index.php/Screenshots den 28 Maj 2007.
- Dictionary.com [WWW document] URL http://www.dictionary.com, sökord: Operation System den 1 Juni 2007.
- GNOME (2004). *GNOME Human Interface Guidelines*. [WWW document] URL http://developer.gnome.org/projects/gup/hig/ den 14 Maj 2007.
- GNOME (2007a). *The Free Software Desktop Project* [WWW document] URL http://www.gnome.org/about/ den 15 Maj 2007.
- Meyer, B. (2005). *Type Manager* [WWW document]. URL http://www.icefox.net/articles/typemanager.php den 25 Maj 2007.
- Microsoft Corporation (2007). *Windows User Experience Guidelines*. [WWW document] URL http://www.microsoft.com/downloads/details.aspx?familyid=B996E1E7-A83A-4CAE-936B-2A9D94B11BC5&displaylang=en den 11 Maj 2007
- Nielsen, J. (1993a): *Non-Command User Infterfaces*. [WWW document] URL http://www.useit.com/papers/noncommand.html den 27 Maj 2007.
- OLPC (2007a). *OLPC Human Interface Guidelines/The Laptop Experience/Zoom Metaphor* [WWW document]. URL http://wiki.laptop.org/go/OLPC_Human_Interface_Guidelines/The_Laptop_Experience/Zoom_Metaphor den 16 Maj 2007.
- OLPC (2007b). OLPC Human Interface Guidelines/Design fundamentals [WWW document]. URL http://wiki.laptop.org/go/OLPC_Human_Interface_Guidelines/Design_Fundamentals#Know_Your_Audience den 17 Maj 2007.
- Raskin, J. (2006a). *Who We Are* [WWW document] URL http://rchi.raskincenter.org/index.php?title=Who_We_Are den 9 Maj 2007.
- Raskin , J. (2006b). *Core Principles* [WWW document] URL http://rchi.raskincenter.org/index.php?title=Core_Principles den 9 Maj 2007.
- Raskin, J. (2006c). *The Zooming User Interface Demo* [WWW document] URL http://rchi.raskincenter.org/index.php?title=Demos#The_Zooming_User_Interface_Demo den 9 Maj 2007.
- SCB (2006). *IT bland individer – Statistik från SCB* [WWW document] URL http://www.scb.se/IT0102 den 28 Maj 2007.
- Tognazzini, B. (2003). *Tog on Interface*. http://www.asktog.com/basics/firstPrinciples.html den 28 Maj 2007.