


Usability Matters!

NUMMER 3
HÖSTEN 1995

Verktyg för design och konstruktion av användargränssnitt är ett område som många forskare och praktiker har intresserat sig för. I det här numret ger vi en översikt av olika typer av verktyg (sid 3) och berättar om två exempel på den svåra avvägningen mellan att utnyttja verktygets möjligheter och styras av dess begränsningar (sid 8). Slutligen beskriver vi olika möjligheter att försörja en systemutvecklare med designkunskap och berättar om vår egen forskning på området kommenterande stödsystem (sid 14). 

Verktyg för gränssnittsdesign . . 3 Utnyttja verktygen! . . 8 Att stödja design . . 14

Nyhetsbladet utges av forskargruppen Usability Matters (UM) vid institutionen för datavetenskap, Linköpings universitet. Ansvarig utgivare är Jonas Löwgren.

UM består av gruppledaren Jonas Löwgren och doktoranderna Torbjörn Näslund, Pär Carlshamre, Mikael Ericsson och Stefan Holmlid. Vår adress är:

Institutionen för datavetenskap, Linköpings universitet, 581 83 Linköping.

Telefon (vx): 013 - 281000. Fax: 013 - 142231.

Email: {jlo, tor, parca, miker, steho}@ida.liu.se

WWW: <http://www.ida.liu.se/labs/aslab/groups/um/>

Om du känner någon som vill ha Usability Matters! i fortsättningen är vi tacksamma för att få veta det. Hör också av dig om du inte själv vill ha flera nummer.

Skicka Usability Matters! till nedanstående person.

Jag har bytt adress. Den nya står nedanför.

Jag vill inte ha fler nummer av Usability Matters!

Namn: _____

Adress: _____

**Faxa till Torbjörn Näslund, 013 - 142231
eller skicka email till tor@ida.liu.se**

Generellt sett är ett verktyg något som stödjer, underlättar eller möjliggör en arbetsuppgift; en ”förlängning” av handen, ögat eller örat. Vilka möjligheter finns då att stödja uppgiften att utveckla användbara system?

Ett sätt att besvara frågan är förstås att titta lite närmare på de aktiviteter som ingår i användbarhetsarbetet. Vid studier av användare och deras arbete förekommer notationer och administrativa stöd för dokumentation av studiens resultat. För prototyping finns en hel mängd verktyg, allt ifrån sax och klisterburk till generella programmeringsspråk. Användbarhetstester kan stödjas med verktyg för datainsamling och -analys. Vissa CASE-verktyg har som ambition att stödja processen i sin helhet. Och så vidare.

I det här numret av Usability Matters! vill vi koncentrera oss på verktygsstöd för användbarhetsorienterat skapande av datorsystemet, och främst sådana verktyg som stödjer skapande av användargränssnittet. I rutan på nästa sida finns lite litteratur om verktyg för andra aktiviteter.

Längre fram i tidningen följer artiklar som diskuterar olika aspekter på verktyg i bruk. I denna inledande artikel ger vi en översikt och lite terminologi som brukar användas inom området.

Typer av verktyg

Man brukar skilja mellan några olika klasser av verktyg för design och konstruktion av användargränssnitt.

Prototypingverktyg stödjer skapande av prototyper som ser ut och beter sig som det önskade systemet men som inte är konstruerade med leverans kvalitet eller i målmiljön. Det är ofta lätt att skapa dynamiska prototyper utan egentlig programmering, och möjligheterna till variation brukar vara mycket stora. Multimedia-verktyg, som t ex Director, används ofta som prototypingverktyg.

Verktyg för gränssnittsdesign

Jonas Löwgren

För att kunna arbeta effektivt gäller det att välja verktyg efter behov. Här ger vi en översikt av olika typer av verktyg, och diskuterar deras för- och nackdelar.

Verktyg för analys och testning

Inom systemutvecklingsområdet finns mängder av notationer för användar- och arbetsstudier, och även CASE-verktyg som administrerar dem. Standardboken om CASE-verktyg är McClure, C. (1989) *CASE is software automation*. Prentice Hall.

Exempel på verktyg för hantering och analys av användbarhetstester beskrivs i Harrison, B. (1995) *Multimedia tools for social and interactional data collection and analysis*, i Thomas, P. (red) *The social and interactional dimensions of human-computer interfaces*, sid 204-239, Cambridge University Press.

Gränssnittseditorer innehåller typiskt en grafisk ritlåda med tillgång till de objekt (knappar, menyer, osv) som ingår i fönstersystemet. Man kopplar ihop gränssnittet med det underliggande systemet genom att automatgenerera gränssnittskod och manuellt bygga ihop den med den övriga koden. Det är ofta mycket lätt att rita gränssnittets statiska delar och sätta objektens attribut, men svårt att bygga dynamiskt beteende. Ett välkänt exempel på en gränssnittseditor är *Developer's Guide*.

4G-verktyg kan beskrivas som programmeringsmiljöer för speciella klasser av tillämpningar, oftast administrativa. De innehåller typiskt databashanterare, rapportgenerator o s v. Många 4G-verktyg idag erbjuder även en integrerad gränssnittseditor, som man kan använda för att utveckla ett grafiskt gränssnitt till den information som lagras i databasen.

Gränssnittshanterare, eller UIMS (user interface management systems), innehåller dels en specifikationsdel där man beskriver gränssnittet och dels en runtime-del som kör gränssnittsbeskrivningen ihop med resten av systemet. Det gör att man får mycket större möjligheter att bygga dynamiska gränssnitt, där objekt skapas, förändras och manipuleras under körningen. Forskningen har kommit ganska långt, men det finns inga särskilt etablerade kommersiella gränssnittshanterare.

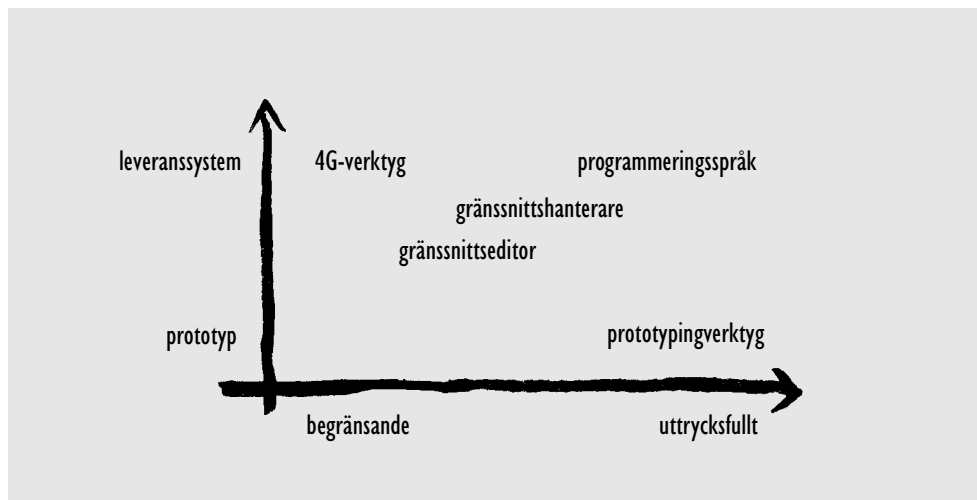
Ett enkelt exempel kan belysa skillnaderna mellan de olika verktygsklasserna. Säg att vi vill bygga gränssnittet till ett CAD-system. I ett prototyping-verktyg skulle vi förmodligen simulera systemets beteende genom att rita en sekvens av skärmbilder. När användaren trycker på en knapp så växlar skärmbilden och det ser ut som om en funktion utförs. En typisk gränssnittseditor skulle bara medge att vi ritade menyer, ikoner och liknande; att visa en CAD-ritning i arbetsytan skulle kräva vanlig programmering i det underliggande systemet. Ett 4G-

verktyg skulle förmodligen inte kunna användas alls, eftersom CAD-system inte tillhör den typ av applikationer som verktyget stödjer. En gränssnittshanterare kunde användas för att beskriva hur ritningens datastrukturer skulle presenteras i arbetsytan, t ex med hjälp av deklarativa samband. Vid exekvering skulle gränssnittshanterarens runtime-del svara för att sambanden hölls giltiga, och om användaren flyttade en komponent i ritningen skulle datastrukturen automatiskt uppdateras.

Egenskaper

Ett verktyg för gränssnittsdesign och -konstruktion kan beskrivas på flera sätt. Tekniska möjligheter är alltid intressanta: grafisk miljö, operativsystem, kompatibilitet med programmeringsspråk, etc. Sådana frågor kommer vi inte att behandla här, utan hänvisar till andra publicerade översikter (se läslistan på sid 7). Vi vill istället koncentrera oss på lite mera allmänna frågor: Stödjer verktyget övergången från prototyp till leveranssystem? Begränsar det möjligheterna till variation och flexibilitet i resultatet, dvs hur uttrycksfullt är det? Diagrammet på nästa sida visar hur de olika typerna av verktyg fördelar sig.

Ett prototypingverktyg är typiskt mycket uttrycksfullt, i bemärkelsen att det medger stor variation i typen av gränssnitt som kan skapas. De flesta verktyg som används för prototyping använder inga standardbibliotek och kräver inte någon viss grundarkitektur. Detta är tveeggat: å ena sidan är det en stor fördel att utforska olika designmöjligheter grundligt, å andra sidan kan det finnas en risk att de lösningar man koncentrerar sig på är svåra att implementera i leveransmiljön. Användning av prototypingverktyg kräver alltså mycket god kännedom om implementeringsarbetets möjligheter och begränsningar.



De olika klasserna av verktyg har olika egenskaper. Vanligen hänger de två dimensionerna ihop: ju mer uttrycksfullt ett verktyg är, desto sämre stödjer det typiskt övergången från prototyp till leveranssystem. Det beror på att i leveranssystemet kan man ofta inte använda innovativa interaktionstekniker.

En gränssnittseditor brukar vara begränsande genom att bara erbjuda standardkomponenter ur en viss grafisk miljö. Dessutom begränsar den indirekt arbetet genom att lämpa sig bäst för statiska gränssnitt; det finns alltså en risk att verktyget styr in designern på formulär- och menybaserade lösningar, trots att mera dynamiska ideer som t ex direktmanipulation kunde varit på sin plats. Gränssnittseditorn stödjer övergången till leveranssystem genom att generera kod, som i bästa fall är av god kvalitet.

4G-verktygen är typiskt mycket begränsande, eftersom de dels stödjer bara en viss typ av applikationer och dels bara erbjuder statiska standardkomponenter för gränssnittet. Däremot är stödet för övergång till leveranssystem mycket bra, eftersom gränssnittet redan vid utveckling är integrerat med den underliggande databasen.

Gränssnittshanteraren är ofta något begränsande genom att den bara stödjer vissa typer av interaktion och gränssnitt. Dock är den mera uttrycksfull än en gränssnittseditor genom att designern kan specificera mera dynamiska gränssnitt, genom att det ofta finns inbyggt stöd för ångra-funktioner och liknande. Övergången till leveranssystem stöds mycket väl i teorin

genom gränssnittshanterarens runtime-del. I praktiken är det väl mera tveksamt om tanken är förenlig med andra krav på kodkvalitet och runtime-miljö. Det kan också vara ett av skälen till att det finns så få kommersiella gränssnittshanterare.

I diagrammet finns generella programmeringsspråk inlagda som jämförelse. Naturligtvis är det så att programmeringsspråket ger störst uttrycksfullhet, och om det stämmer med målmiljön ger det också idealt stöd för utveckling av leveranssystem. Men priset man får betala är att programmeringsspråkets abstraktionsnivå inte lämpar sig särskilt väl för gränssnittsdesign och att användargränssnitt kan vara komplicerade att programmera. De verktyg vi behandlar här ger bättre stöd för arbetet.

Trender

En hel del arbete pågår kring frågan om andra typer av stöd för design och konstruktion av användargränssnitt. Till exempel har man intresserat sig för designerns försörjning av designkunskap, speciellt sådan lågnivåkunskap som normalt finns i skrivna designregler, plattformsspecifika eller företagsspecifika "style guides". Ett annat exempel på utökat stöd är notationer och verktyg för att dokumentera designerns argumentation kring designbesluten, s k design rationale.

Inom området gränssnittshanterare verkar trenden gå i riktning mot mera integrerade miljöer för applikationsutveckling, där användargränssnittet tidigt kommer i fokus.

Läs mera

När det gäller verktyg för design och konstruktion av användargränssnitt finns det sammanställningar att få tag på, till exempel Bergsten, P. m fl (1993) *Verktyg för grafiska användargränssnitt*. SISU-rapport nr 20, SISU, Electrum 212, 164 40 Kista. Den utmärkta Web-samlingen HCI Resources innehåller också verktygsinformation (<http://www.ida.liu.se/~miker/hci/>, leta under "Software").

Mycket av den aktuella forskningen om gränssnittshanterare sammanfattas i ett par översiktsartiklar: Olsen, D. m fl (1993) *Research directions for user interface software tools*, Behaviour & Information Technology 12(2):80-97, och Myers, B. (1995) *User interface software tools*, ACM Trans. Computer-Human Interaction 2(1):64-103.

Utnyttja verktygen!

*Pär Carlshamre
Torbjörn Näslund*

Det är inte lätt att dra nytta av ett verktyg utan att bli hämmad av verktygets begränsningar. Vi visar två exempel på detta.

Att använda verktyg för prototyping och datasystemkonstruktion är oftast tidsbesparande. En mycket stor del av den besparing som man gör i tid beror på att verktygen tillhandahåller schablonlösningar för vanliga konstruktionssituationer. Med en liknelse kan säga att verktygen tillhandahåller ”halvfabrikat”, som förhållandevis snabbt kan sättas ihop till en prototyp eller ett fungerande datasystem. Som systemdesigner kan man många gånger gå upp ett steg i abstraktionsnivå, och mer bry sig om vad datasystemet skall göra i stora drag, medan verktyget effektivt ger stöd för detaljerna i konstruktionen.

Precis som vid all användning av halvfabrikat finns det dock en motsättning mellan snabb utveckling och flexibilitet. Vi köper den ökade snabbheten i utvecklingen genom att offra en del flexibilitet. Många gånger är detta ett bra val. Den minskade flexibiliteten kan bland annat medföra att vi får en ökad standardisering av våra datasystem. I andra fall kan dock den minskade flexibiliteten leda till att vi får svårt att utveckla exakt det system som vi skulle vilja. Vi blir styrda av verktyget.

Det gäller alltså att finna en balans, där vi kan dra nytta av verktyget, men där vi inte blir alltför bakbundna av de alternativ som verktyget ger. I denna artikel ger vi två verkliga exempel på situationer där verktygets begränsningar har haft stor betydelse.

I vårt första exempel illustrerar vi hur systemutvecklare i ett stort systemutvecklingsprojekt alltför lättvindigt anpassade sig till de standardlösningar som utvecklingsverktyget föreslog. Trots att verktyget gav möjlighet till andra, och mer lämpliga, lösningar, valde systemutvecklarna att mer eller mindre medvetet styras av verktyget.

I vårt andra exempel visar vi hur en grupp teknologer utmanade de begränsningar som ett prototypingverktyg satte. De lyckades därmed töja på de gränser som verktyget egentligen satte.

Första exemplet: Standardlösningar

I en av våra fallstudier i professionell systemutveckling hade systemutvecklarna bestämt sig för att använda ett databasorienterat verktyg. Verktyget kan beskrivas som ett 4G-verktyg med stor flexibilitet för utformningen av användardialogen. Verktyget byggde på en egen tankemodell för hantering av stora mängder data. Tankemodellen gick i stort ut på att användaren valde ut delmängder av den totala datamängden, och sedan navigerade fram och tillbaka inom den utvalda delmängden. Verktyget var utformat så att denna tankemodell i normalfallet också slår igenom för de datasystem som utvecklas med verktyget.

En rimlig bedömning är att en användare skulle kunna lära sig denna tankemodell efter en endagskurs och några timmars övning.

Det datasystem som skulle utvecklas i just detta projekt hade dock speciella krav: Det måste gå mycket snabbt att lära sig använda systemet. Systemet måste i stort sett vara helt intuitivt att använda. Däremot skulle det sällan bli fråga om några större datamängder för användaren att hantera. För just detta datasystem var verktygets normala tankemodell olämplig, och alltför svår att lära sig.

Systemutvecklarna i projektet lärde sig naturligtvis snabbt den underliggande tankemodellen, eftersom de arbetade med den dagligen. De blev därmed också snabbt omedvetna om att många av de designöverväganden som de gjorde baserade sig på denna underliggande, men alltför komplexa, tankemodell.

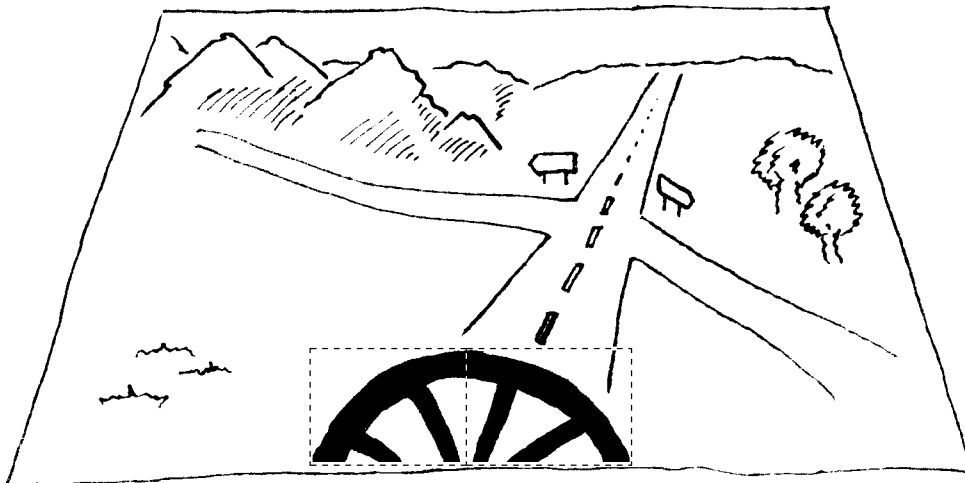
Systemutvecklarna följde i stor utsträckning de standardlösningar och standardalternativ som verktyget föreslog. Deras uppfattning var att verktygets standardlösningar sannolikt var väl genomtänkta och väl utprovade. De uppfattades också som systematiskt genomförda.

Dessvärre gick systemutvecklarna i en fälla. Verktygets standardlösningar var anpassade för *en annan typ* av datasystem än det som just nu var aktuellt. Verktyget var i och för sig användbart även för den aktuella data-tillämpningen, men krävde då att systemutvecklarna inte ”följde den snitslade banan”, utan själva valde bland, och anpassade, de möjligheter som verktyget erbjöd.

Andra exemplet: Att sträcka gränserna

Vi ger en kurs i människa-datorinteraktion för teknologer här på Linköpings universitet. I kursen lägger vi stor vikt vid kreativitet, och ger studenterna stor frihet att experimentera. Detta sker bland annat med hjälp av lofi-prototyping, där man använder sig av papper, tejp, olivfärgade pennor m m för att illustrera en idé. Dessa lofi-prototyper brukar typiskt vara mycket vidlyftiga, då man verkligen har lagt sig vinn om att skapa något som sticker ut från det ordinära.

Problemen dyker upp när studenterna sedan skall överföra prototyperna till exekverbar form. Det enda verktyget som stått till buds har hittills varit Developer's Guide för Sun/OpenWindows, vilket är en tämligen enkel gränssnittseditor. DevGuide stödjer t ex inte generell drag-and-drop, vilket direkt utesluter direktmanipulation av gränssnittsobjekt. Det stödjer heller inte dynamik i större utsträckning; man kan inte få saker att röra sig på skärmen. Efter den kreativa lofi-prototypingen brukar således mötet med DevGuide upplevas som ganska smärtsamt av studenterna.



Till vår glädje upphör vi dock aldrig att förvånas över hur studenterna lyckas kringgå dessa begränsningar, och med risk för att bli alltför teknisk vill jag ge ett par exempel på detta.

DevGuide ger möjlighet att svara på vissa händelser (t ex musklick) genom att "visa" respektive "gömma" olika objekt. Detta användes av ett par studenter för att simulera en ratt som man kunde vrida på för att navigera i en virtuell omgivning (se skissen ovan). Man skapade helt enkelt ett stort antal ikonpar, som representerade vänster resp. höger sida av ratten. För varje par var ratten vriden några grader åt ena eller andra hållet. Om man klickade på vänster sida av ratten så "vreds" den åt vänster genom att man "gömde" ikonerna" och "visade" nya ikoner på samma ställe, men med en annan lutning på ratt-ekrarna. Genom att hålla nere musknappen kunde man vrida ratten olika långt. Samtidigt ändrades bilden av omgivningen (dvs det användaren såg genom "vindrutan") på samma sätt genom att omväxlingsvis gömma och visa bilder. Man hade således lyckats åstadkomma ett mycket dynamiskt beteende med ett verktyg som nästan inte alls stödjer sådant.


Med hjälp av ratten kan man navigera i det virtuella informationslandskapet genom att svänga till vänster eller höger i vägkorsningarna.

DevGuide ger också möjlighet att under vissa omständigheter byta utseende på pekarmarkören till en godtycklig bitmap. Detta utnyttjades i ett annat fall för att simulera drag-and-drop. Vad som händer visuellt när man med musen drar ett objekt från sin plats och släpper det någon annanstans på ytan är att det först signalerar att det blivit markerat, därefter följer det med musen över ytan, sedan får det en droppytta att signalera (t.ex. genom att den blir mörk) att den är mottaglig för objektet, varpå slutligen objektet fastnar på droppytan då man släpper musknappen. Enkelt, tänkte teknologerna, och lät ett objekt ”gömmas” då man klickade på det och höll ned musknappen. Samtidigt lät man pekarmarkören byta form till en pil ovanpå ett likadant objekt, som alltså kunde dras över ytan. För att simulera att droppytan var mottaglig för objektet var man visserligen tvungen att släppa musknappen och klicka en gång till på droppytan, men det fungerade tillräckligt väl för att illustrera idén för användaren. Slutligen sa man åt ett nytt objekt att ”visa” sig i droppytan, för att signalera att det nu hade förflyttats dit.

Dessa exempel visar att det ofta är fullt möjligt att kringgå de begränsningar som ett verktyg medför. Man undrar förstås om det var värt besväret. ”Vi skulle aldrig ha tid att hålla på med sånt där” kanske någon säger. Man skall då vara medveten om att teknologerna i detta fall haft två timmar på sig att bekanta sig med ett nytt verktyg — i de flesta fall en helt ny typ av verktyg — och sedan haft fyra timmar på sig att skapa sin körbara prototyp. Den slutsats som vi drar måste i alla fall bli att man bör vara ganska tveksam när man hör någon säga att ”Nä, det där kan man inte göra med det här verktyget.”

Råd för verktygsanvändning

Utgående från de två exemplen vill vi ge följande allmänna råd för verktygsanvändning vid användbarhetsarbete.

- *Gör en medveten design av ditt system.* Ta hänsyn till vad dina verktyg ger möjligheter till, men låt inte designen styras av dina verktyg.
- *Var beredd att avstå från designmöjligheter som verktyget ger.* De kanske inte alls är lämpliga för det system du håller på att bygga.
- *Utnyttja den flexibilitet som verktyget erbjuder,* även om det kan ge merarbete jämfört med att hela tiden följa verktygets standardrekommendationer. Du har sannolikt i alla fall sparat mycket tid på att använda verktyget, jämfört med att programmera utan verktyg! 

Att stödja design

Mikael Ericsson

Gränssnittsdesign kräver kunskap. Man kan tänka sig att verktygen stödjer arbetet på nya sätt.

Tänk om verktyget du använder för att skapa gränssnitt inte bara gav stöd för design och implementation, utan också kunde hjälpa dig utvärdera resultatet. Tänk dig att du fick stöd för att hantera problemen med att dels hålla dig inom riktlinjerna ni upprättat för projektet, men också med att följa de generella standarder ni måste följa för att få nästa projekt. Ett scenario: Under arbetet med en komplex design av ett informationssystem är det av yttersta vikt att terminologin i fält och ledtexter är konsekvent. Ditt verktyg gör dig uppmärksam på tveksamheter genom att peka ut objekt som avviker. Eftersom kunden finns i ett annat EU-land är du tvungen att följa de generella riktlinjer för användbarhet som de nationella standarderna föreskriver, samtidigt som du som vanligt måste följa Motif style guide. Verktyget pekar ut eventuella avvikelser eller tveksamheter.

Behov av stöd

De flesta verktyg för arbete med gränssnitt, och för mjukvara i allmänhet, ger stöd för att skapa och modifiera system. I många fall är det dessutom enkelt att bygga hela applikationer, genom att ”dra” byggblock från en objektpalett, skapa det visuella gränssnittet, och automatiskt generera kopplingen till de underliggande funktionerna; 4G-verktyg är bra exempel på detta.

Förutom att skapa applikationer är det minst lika viktigt att kunna utvärdera och omdesigna dessa. Detta är något som inte stöds i större utsträckning av dagens verktyg, trots att det är lätt att se behovet av stöd: Det blir allt viktigare att kunna testa sina applikationer med avseende på användbarhet, konformitet med plattformspecifika och företagsspecifika riktlinjer eller regler för produktidentitet och samtidigt följsamhet mot nationella/internationella standarder (t.ex. ISO9241).

De dokument som innehåller riktlinjer och standarder är alltför många, alltför stora, allför generella ("var konsekvent") och i många fall inkonsekventa eller motsäggande. Sammantaget gör detta dem svåra att använda. Den svårhanterliga mängden kunskap som riktlinjerna innehåller och det faktum att de riskerar att förändras/ anpassas mellan olika projekt gör det svårt att hålla sig a jour med vad som skall tillämpas för stunden. Kort sagt — kunde man på något sätt få stöd för hanteringen av riktlinjer skulle man dels lösa resursproblem, dels få stöd för att följa regler och externa krav. I många fall skulle man dessutom öka möjligheten för egen kritisk granskning av designen.

Olika typer av stöd

Man har prövat många sätt att angripa problemet.

Online-dokument. För att göra det lättare att hitta information kan man tillhandahålla riktlinjer i hyper-textform eller i en databas och därmed ge utvecklaren bättre möjligheter att söka information under arbetet.

Exempeldatabas. Det är inte bara svårt att hitta information; det handlar också om att försöka se var denna är relevant och på vilket sätt. Genom att förse riktlinjerna med exempel i form av illustrationer och scenarier kan man ge stöd för detta.

Begränsningar via toolkits. Rent tekniskt sett är det lätt att råka skapa funktioner och objekt som inte överensstämmer med riktlinjerna. De toolkits och kodbibliotek som används kan utformas så att det blir svårt eller till och med omöjligt att göra sådana missar.

Begränsningar via widgets. I det fall arbetet sker i en gränssnittseditor kan samma "begränsningar" överföras via t ex widgetpaletten: samtliga widgets är anpassade efter riktlinjerna och kan inte fås att avvika från dessa. Många av dagens verktyg ger denna typ av stöd när det gäller plattformsspecifika riktlinjer.

Riktlinjer för gränssnittsdesign

Med riktlinjer avser vi de rekommendationer, råd och regler som återfinns i dokument som t ex *Guidelines for designing UI software* av Smith och Mosier, *Motif style guide* eller *Macintosh Human Interface Guidelines*. Några exempel på fraser ur sådana dokument:

- Var konsekvent i terminologi och förkortningar.
- När data används i spatial eller temporal ordning, gruppera i användningsordning.
- Då användaren matat in ett otillåtet värde i ett fält skall ett felmeddelande visas direkt. Detta skall innehålla information om tillåtna värden (intervall, format, etc.).

Ordlista

toolkit: ett programmeringsbibliotek med fördefinierade procedurer för att implementera interaktionsobjekt.

widget: interaktionsobjekt, som knappar och inmatningsfält. Även en större enhet som en filvalsdialog kan vara en widget.

guidelines: generella regler för god gränssnittsdesign, ofta baserade på experimentell forskning inom MDI-området.

style guide: en samling designregler för användargränssnitt i en speciell miljö, t ex Windows, Mac eller Motif.

Mallbibliotek. Företags-, produkt-, eller projektspecifika riktlinjer kan på samma sätt som de plattformspecifika i viss mån stödjas genom implicita begränsningar: verktygen gör det möjligt att bygga upp bibliotek av gränssnittskomponenter (komplexa widgets och grupper av widgets) och återanvända dessa.

Konsekvenskontroll. Andra riktlinjer, som rör t ex visuell konsekvens eller terminologi, kan stödjas genom funktioner i verktygen. Placerings- och grupperingsfunktionen innehåller mekanismer för att automatiskt justera objekt i förhållande till andra. Text stavningskontrolleras och stäms av mot terminologidatabas.

Automatdesign. Liknande stöd kan förmedlas med verktyg som utför automatisk design. Systemutvecklaren specificerar funktioner och objekt i någon formell notation. Verktyget genererar ett eller flera gränssnittsförslag, som sedan kan justeras.

Kommenterande system. Om man vill låta systemutvecklaren utföra all layout och handgripligt arbete med gränssnittet, reducera arbetet med att hitta riktlinjer, men ändå ge stöd för användningen av dessa, kan man använda kunskapsbaserat stöd och kommenterande verktyg. Ett expertsystem utvärderar gränssnittet och ger utvecklaren kommentarer som bygger på riktlinjer.

I samtliga dessa ansatser måste man dessutom ta med i beräkningarna att riktlinjer i många fall ”krockar” med andra krav och mål i ett designprojekt. Dessutom, som kreativ designer vill man kanske kunna avvika från gängse riktlinjer ibland. Ett verktyg som ger stöd för hanteringen av riktlinjer bör därför kunna anpassas mellan och under projekt.

Vår forskning kring verktygsstöd

I forskningsprojektet Codek undersöker vi möjligheten att använda kunskapsbaserat stöd, i form av ett kommenterande expertsystem, för hantering av riktlinjer i

en omgivning för gränssnittsdesign. Man vet att det är tekniskt möjligt att utföra utvärdering och granskning av gränssnitt med hjälp av expertsystem, åtminstone för vissa typer av riktlinjer. Vad som inte varit känt tidigare är om ”stöd” av denna typ är användbart. Vill man som systemutvecklare arbeta med en kommenterande ”agent”? Blir man mer effektiv med detta stöd? Blir det resulterande gränssnittet bättre? Det är dessa frågor som vi försöker besvara genom att empiriskt studera användningen av ett kommenterande system under arbete med en designuppgift.

Vi använder så kallade Wizard-of-Oz-experiment i projektet, vilket innebär att vi inte implementerat ett riktigt kommenterande expertsystem, utan låter en mänsklig expert simulera detta. Under försöket låter vi dock försökspersonerna tro att det är ett riktigt system (efteråt berättar vi givetvis hur det egentligen ligger till). Detta ger oss möjligheten att testa ”systemet” under flera olika betingelser, utan att behöva lägga stora resurser på implementation. Vårt nuvarande system består av en enkel gränssnittseditor, ett simulerat kommentarverktyg och en miljö för att övervaka och presentera kommentarer för försökspersonen.

Vi är primärt intresserade av huruvida ansatsen är användbar, och vilka strategier som i så fall är bäst, d v s på vilket sätt kommentarer skall ges. Är ett *aktivt* system (kommenterar omedelbart) bättre än ett *passivt* (kommenterar på begäran)? Är *deklarativa* kommentarer (pekar ut avvikelser) bättre än *imperativa* (beskriver alternativ design)? Dessutom studerar vi *inlärningseffekter* av användningen av ett kommenterande system: lär man sig något om riktlinjer under tiden?

Våra preliminära resultat visar att ett kommenterande system av användaren uppfattas som nyttigt, men samtidigt störande. Detta kan verka paradoxalt, men för-


klaras av ett tredje faktum — de flesta angav att kommentarerna innehöll riktlinjer som de inte kände till tidigare. De upplevde alltså att systemet avbryter designarbetet, men uppskattade den nya informationen. Vad gäller olika strategier visade det sig att:

- Kommentarer som beskriver vad som skall göras för att åtgärda ett problem är lättare att förstå än kommentarer som bara pekar ut problemen.
- Kommentarer som presenteras av ett aktivt system riskerar att passera obemärkta om designern är hårt mentalt belastad. Detta är i sig inget konstigt, men utgör en viktig faktor inför konstruktionen av en riktig kommenterande agent: för att systemet skall ”nä fram” till designern måste det ha kunskap om huruvida denne för stunden är mottaglig eller ej.

För tillfället är vi sysselsatta med närmare analyser av resultaten, och planerar att kunna presentera slutsatser och iakttagelser före nyår. Våra långsiktiga planer är att utveckla en prototyp baserat på resultaten av våra försök och testa den i en industriell utvecklingsmiljö.

Den komplexa verkligheten

Dagens tillvaro är egentligen mer komplex än vad som målats upp ovan. Förutom att det saknas stöd för att hantera de riktlinjer som ställs upp idag dyker det hela tiden upp nya krav på verktygen. Som designer vill man kunna skapa helt nya interaktionsobjekt: det räcker inte med traditionella knappar, rullningslistor eller menyer — produkten kan t ex kräva animerade objekt med strömlinjeformer. Finns stöd för att skapa VR-gränssnitt? Här blir också möjligheten att avvika från riktlinjer mycket viktig: systemutvecklaren bör ges möjlighet att selektivt koppla bort riktlinjer.

Mycket återstår att göra när det gäller designstödsverktyg, men Codek-projektet är ett steg på vägen. 

Carlshamre, P. och Tumminello, J. (1995). Technical communicators' views on usability and collaboration.

Fem teknikinformatörer i USA och lika många i Sverige intervjuades om sin arbetsituation, speciellt användbarhetsarbete och samarbete med systemutvecklare. Resultaten visar att teknikinformatörer är värdefulla användbarhetsresurser, men hindras av sin egen och kundernas organisationer. Situationen är intressant nog densamma i USA och i Sverige.

Carlshamre, P. (1994). A collaborative approach to usability engineering: Technical communicators and system developers in usability-oriented systems development.

En utförlig rapport från Delta-projektet, där vi studerade införande av användbarhetsorienterade tekniker och samarbete mellan systemutvecklare och teknikinformatörer i professionell systemutveckling. Pär studie visar på goda möjligheter till användbarhetsarbete i praktiken, diskuterar några kritiska problem och visar hur de kan undvikas.

Löwgren, J. (1995). Perspectives on usability.

I den vetenskapliga MDI-litteraturen kan man finna olika sätt att se på begreppet användbarhet. Rapporten går igenom några olika synsätt och visar vad de får för konsekvenser för systemutveckling.

Löwgren, J. (1995). Applying design methodology to software development.

Mer och mer bevis börjar samlas för att de modeller som ofta används idag för att styra systemutveckling inte stämmer särskilt väl överens med vad som faktiskt händer i projekten. Artikeln diskuterar en modell från design teorins område och visar vad det skulle innebära om man beskrev systemutveckling utifrån modellen.

(forts på pärmens baksida)

Mer att läsa

Vi dokumenterar naturligtvis vår forskning i vetenskapliga publikationer. Här är ett aktuellt urval.

Om du vill ha kopior av våra skrifter så kontakta respektive författare eller Birgitta Franzén,
tel: 013 - 282692,
email: birfr@ida.liu.se

Löwgren, J., Carlshamre, P. och Näslund, T. (1995). Developing and studying usability-oriented methods in professional contexts.

Den här artikeln är en sammanfattning av vår praktikinära forskningsfilosofi med två exempel: samarbete mellan teknikinformatörer och systemutvecklare i Delta-projektet, samt hur användbarhetsinspektioner fungerar i iterativ utveckling.

Näslund, T. (1995). Computers in context — but which context?

Artikeln bygger på en fallstudie från ett systemutvecklingsprojekt i näringslivet. Fyra grupper av projektdeltagare identifieras, var och en med sitt eget sätt att se på systemutvecklingen. Skillnaderna innebär att deltagarna kan fokusera på de aspekter som de är speciellt kunniga på, men medför också en risk att missförstånd uppstår.

Näslund, T. (1994). "Usability is extremely important — but it's somebody else's job, I hope".

Läroböcker i människa-datorinteraktion poängterar ofta vikten av att datasystemen blir användbara. Däremot är bilden en annan inom systemutveckling och software engineering. Systemutvecklingslitteraturen förutsätter typiskt att användbarhet ska ordnas vid programkonstruktionen, men ger inte många råd om hur det ska gå till. I böckerna om programvaruproduktion förutsätts i stället att användbarhetsfrågorna är utredda innan produktionen startar.

Nästa nummer kommer under våren 1996.

Det kommer att handla om centrala roller i användbarhetsarbete.