

Komunikacja Człowiek-Komputer

Wojciech Miksa



Laboratorium 2: Graficzny interfejs użytkownika.

Laboratorium/projekt



SPOŁECZNA AKADEMIA NAUK

GRAFICZNY INTERFEJS UŻYTKOWNIKA

- Pierwsze interfejsy pojawiły się jeszcze w latach 60-tych, początkowo z piórem świetlnym.
- Pierwszy interfejs graficzny z wykorzystaniem myszki opracował XEROX w latach 70-tych.
- W Polsce pierwsze GUI były w komputerach Odra 1204 w latach 70-tych (24 bit, obliczenia równoległe).



GRAFICZNY INTERFEJS UŻYTKOWNIKA

- Komputery Apple spopularyzowały GUI w latach 80 na rynku konsumenckim. Wiele innych komputerów miało opracowany interfejs graficzny:
 - Commodore 64 (GEOS – jako opcjonalny program)
 - Amiga (Workbench)
 - Atari ST (TOS)
 - PC (OS/2, Windows 3.1)
- Na rynku PC, dopiero od premiery Windows 95, GUI stał się standardem.



ZALETY

- Ogromna łatwość nauczenia się użycia interfejsu – nie wymaga zapamiętywania magicznych formuł (komend).
- Możliwa jest bezpośrednia interakcja z prezentowanymi elementami modelu rzeczywistości, np. punktami na mapie, plikiem na dysku twardym czy fragmentem części w rysunku CAD.



ZALETY C.D.

- Możliwe jest przejrzyste prezentowanie informacji.
- Jest atrakcyjny wizualnie.
- GUI można uzupełnić wpisywaniem tekstu.



WADY

- Większe zapotrzebowanie na pamięć, moc procesora, w konsekwencji na pobór energii elektrycznej.
- Wymagane jest ekran z odpowiednią rozdzielczością, najlepiej kolorowy i kontroler lub ekran dotykowy.



WADY C.D.

- Precyzja klikania jest ograniczona (ma to znaczenie np. w programach CAD – Computer Aided Design lub budowie siatki w obliczeniach MES, gdy trzeba dorysować kreskę zamykającą kontur lub zaczynającą się od punktu czy krawędzi).
- Wielokrotne kliknięcie w obszar okienka jest wolniejsze od wpisania komendy, przez co dla zaawansowanego użytkownika, używanie okienek może być wolniejsze od wpisywania komend.



PROJEKTOWANIE INTERFEJSU UŻYTKOWNIKA

- Norma oceny jakości oprogramowania
 - ISO/IEC 25010 : 2011 “Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models”
 - 4.1 Quality in use – jakość eksploatacji
 - 4.2 Product quality – jakość produktu
- Heurystyki Nielsena. Zestaw 10 reguł jest wynikiem przeprowadzonych badań z 1990 roku.
 - Dalsza część prezentacji jest oparta na heurystykach Nielsena z komentarzem prowadzącego.



1. WIDOCZNOŚĆ STATUSU SYSTEMU

- Użytkownik powinien zawsze widzieć, co się dzieje.
- Do informowania użytkowników służą:
 - wyświetlanie tekstu w okienkach lub paskach (u dołu ekranu),
 - pasek postępu
 - lub klepsydra.



2. DOPASOWANIE MIĘDZY PROGRAMEM A RZECZYWISTOŚCIĄ

- Powinno się używać określeń znanych użytkownikowi, zamiast żargonu technicznego.
- Nie należy zakładać, że inni znają słownictwo, które my sami znamy.



2. DOPASOWANIE MIĘDZY PROGRAMEM A RZECZYWISTOŚCIĄ

- Dopasowanie zależy od przewidywanej grupy docelowej, np.:
 - Prosty język jest właściwy dla dzieci w wieku przedszkolnym.
 - Język bez żargonu i zbyt skomplikowanego słownictwa jest właściwy dla przeciętnych dorosłych.
 - W przypadku osób wykorzystujących specjalistyczne oprogramowanie ma sens stosowanie znanego użytkownikom, specyficznego żargonu.
np. wśród inżynierów mechaników nie mówimy śrubokręt, tylko wkrętak.



3. KONTROLA PROCESU I SWOBODA DZIAŁANIA

- Użytkownikom zdarza się wykonywać akcje przez pomyłkę, dlatego powinni mieć kontrolę nad tym co się dzieje.
- Dla każdej akcji powinna być możliwość wycofania się do wcześniejszego etapu.



3. KONTROLA PROCESU I SWOBODA DZIAŁANIA

- Powinno być zaimplementowane operacje „cofnij” oraz „ponów”.
- Powinny być dobrze widoczne możliwości wyjścia, np. przyciski „wstecz”, „anuluj”.



4. SPÓJNOŚĆ I STANDARDY

- Program powinien zachowywać się wszędzie w podobny sposób, by ułatwić użytkownikowi obsługę.
- Zachowanie programu: nie należy zmuszać użytkownika do uczenia się zbyt wielu rzeczy, stosując nietypowe sposoby reakcji programu.
- Spójność operacji: powtarzające się przyciski (dalej/wstecz, tak/nie) powinny mieć podobny rozkład i być w taki sam sposób nazywane.



4. SPÓJNOŚĆ I STANDARDY

- Spójność wizualna: podobny styl graficzny w całym programie.
- Także w GUI, konwencje mają znaczenie – użytkownicy są przyzwyczajeni do pewnych rozwiązań. np.:
- W modalnym okienku dialogowym przycisk „anuluj” jest po lewej, „Ok” po prawej



5. ZAPOBIEGANIE BŁĘDOM

- Lepiej jest zapobiegać powstawaniu błędów, niż przeciwdziałać ich skutkom.
- Kontrolki nie muszą przyjmować błędnych danych, np. daty urodzenia w niewłaściwym formacie, liter dla numeru PIT itd.



5. ZAPOBIEGANIE BŁĘDOM

- Wskazywanie obiektów jest mniej podatne na błędy niż wpisywanie ich nazw (a jeszcze np. nazwy miejsc czy produktów mogą być zapisywane na różne sposoby, np. Falklands vs Malvinas, MiG-17PF vs Lim-5P vs Shenyang J-5A, migawka vs bilet okresowy).
- Pisanie krótkich wyjaśnień czy wskazówek pomaga użytkownikom zrozumieć działanie programu.



6. ROZPOZNAWANIE ZAMIAST PRZYPOMINANIA

- Krótkoterminowa pamięć człowieka jest ograniczona, użytkownik nie powinien być zmuszony do pamiętania, jak wykonać potrzebną operację.
- Należy ograniczać konieczność zapamiętywania czegokolwiek.



6. ROZPOZNAWANIE ZAMIAST PRZYPOMINANIA

- Najważniejsze opcje menu powinny być widoczne, by nie obciążać pamięci.
- Chociaż dla przejrzystości pozycji nie powinno być zbyt wiele.
- Nad ikonkami czy kontrolkami można umieszczać pojawiające się po najechaniu kursorem podpowiedzi.



7. ELASTYCZNOŚĆ I SPRAWNOŚĆ UŻYCIA

- Możliwość wykonywania operacji na skróty znacząco usprawnia pracę zaawansowanym użytkownikom.
- Skróty klawiszowe (Ctrl +C – skopiuj do schowka, F5 – nagraj stan gry) znacząco przyspieszają wykonanie operacji, w porównaniu z chodzeniem po menu.



7. ELASTYCZNOŚĆ I SPRAWNOŚĆ UŻYCIA

- Personalizacja umożliwia użytkownikowi dostosowanie interfejsu do swojej wizji wygody.
- Np. można zmieniać widoczne ikonki wykonujące operacje lub definiować własne skróty klawiszowe, zamiast grzebania w menu.



8. ESTETYCZNY I OSZCZĘDNY WYGLĄD

- Ilość informacji należy ograniczyć do tych naprawdę i chociaż dość często potrzebnych.



9. ZAPEWNIENIE POMOCY W DIAGNOZOWANIU BŁĘDÓW

- Komunikaty o błędach powinny być zrozumiałe i przydatne.
 - Informacja powinna być w języku zrozumiałym dla użytkownika.
 - Kody błędów nie są zrozumiałe, można je użyć najwyżej w specjalistycznym oprogramowaniu jak IDE.



9. ZAPEWNIENIE POMOCY W DIAGNOZOWANIU BŁĘDÓW

- Przydatne dla użytkownika mogą być podpowiedzi rozwiązań problemu
- Wyskakujące okienka modalne potrafią być irytujące, gdyż trzeba w nie klikać.



10. POMOC I DOKUMENTACJA

- Nawet, jeśli program nie wymaga szczegółowych wyjaśnień, może być przydatny opis typowych zadań, które ma wykonać użytkownik.



10. POMOC I DOKUMENTACJA

- Opis zadań powinien zawierać konkretne kroki do wykonania.
- Można tworzyć tutoriale (ćwiczenia), demonstracje , przykłady.
- Dokumentacja powinna być zwięzła i łatwa do przeszukiwania.



ZADANIE 2

- Celem ćwiczenia jest wykonanie 3 szkiców graficznego interfejsu użytkownika.
- Z założenia szkice powinny dotyczyć jednej i tej samej aplikacji.



NARZĘDZIA

- Zintegrowane środowisko programistyczne (IDE - ale wystarczą same zrzuty ekranu – nie trzeba programować).
 - Visual Studio Community Edition
 - Code Blocks z WX Widgets
 - Eclipse (jednak Java wymaga odrobiny programowania do stworzenia GUI)
- Kod NIE JEST OCENIANY (nie na tych ćwiczeniach)
- Program graficzny:
 - MS Paint.
 - Photoshop



KRYTERIA OCENY

- Wymagane są 3 szkice.
- Na każdym szkicu powinno być minimum 5 kontrolek (przyciski, okna edytowalne, tabele itp., razem co najmniej 15 w pracy).
- Spójność stylu.
- Przejrzystość szkicu.
- Zgodność z konwencjami.



DZIĘKUJĘ ZA UWAGĘ

Wojciech Miksa

wmiksa@san.edu.pl



SPOŁECZNA AKADEMIA NAUK

