**ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ (РІВЕНЬ D - ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ ТА ОЦІНЮВАННЯ ЛЮДИНО - МАШИННИХ ІНТЕРФЕЙСІВ)**

**Лабораторні роботи**

**Лабораторна робота №3**

1. Мета роботи. Створення прототипу інтерфейсу веб-додатку.

2. Методичні вказівки.

При створенні інтерфейсу рекомендується використовувати існуючі принципи проектування користувальницького інтерфейсу. Далі наводяться шість принципів, які увібрали в себе багато чого з того, що на даний момент відомо про розробку ефективного призначеного для користувача інтерфейсу. Кожен з них включає в себе кілька пов'язаних між собою ідей, більш деталізованих в порівнянні з загальними питаннями. Цими загальними питаннями є структура, простота, видимість, зворотний зв'язок, толерантність і повторне використання.

**Структурний принцип**

Організація призначеного для користувача інтерфейсу повинна бути доцільною, осмисленої і зручною. Вона повинна базуватися на чітких, цілісних моделях, очевидних і розпізнаються користувачами. При цьому родинні поняття повинні бути пов'язані, а незалежні - розділені.

Не схожі елементи повинні диференціюватися, а схожі - виглядати схоже. Структурний принцип пов'язаний із загальною архітектурою інтерфейсу і безпосередньо відображає уявлення про користувальницький інтерфейс як про діалог між розробниками і користувачами. Організація хороших інтерфейсів продумується дуже ретельно, таким чином, щоб відображати структуру розв'язуваних системою завдань і спосіб мислення користувачів щодо цих завдань. Дуже часто, особливо при використанні сучасних візуальних середовищ розробки, розташування візуальних компонентів всередині форм або діалогів і їх розподіл між ними виявляється майже випадковим і відображає в кращому випадку послідовність, в якій програмістами порушувалися ті чи інші питання. За ідеєю, властивості і функції, які найчастіше використовуються спільно або розглядаються користувачами як пов'язані один з одним, повинні розташовуватися разом або, принаймні, повинні бути чітко і ясно взаємопов'язані. Що ж до тих елементів, які в контексті завдання або в свідомості користувача ніяк не пов'язані між собою, то вони повинні бути рознесені в інтерфейсі. Подібне має бути подібно. Схожа інформація повинна бути організована за допомогою схожих рішень, а об'єкти, що володіють схожим поведінкою, повинні мати загальне уявлення.

**Принцип простоти**

Слід максимально спрощувати управління найбільш поширеними операціями. При цьому спілкування з користувачем повинно вестися зрозумілою для нього мовою. Повинні надаватися посилання, логічним чином вказують на більш складні процедури. Процес проектування інтерфейсу - це завжди боротьба за компроміс. Спрощення чогось одного неминуче призводить до ускладнення чогось іншого. Якщо зменшити кількість меню, збільшиться число пунктів в кожному з них. Якщо зробити маленькими все діалогові вікна, включивши в них якомога менше елементів, будь-яка взаємодія користувача з системою обернеться для нього необхідністю звертатися до великої кількості таких віконець. Неможливо зробити все на світі простим. Дотримання принципу простоти вимагає від вас знання того, які завдання виконуються користувачем найбільш часто і які з них, з точки зору користувача, простіше. Саме такі завдання слід спрощувати, щоб користувач міг швидко їх вирішити.

**Принцип видимості**

Всі функції і дані, необхідні для виконання даного завдання, повинні бути видні, щоб користувач не відволікався на додаткову і надлишкову інформацію.

Принцип видимості пов'язаний з проектуванням таких користувальницьких інтерфейсів, в яких видно всі елементи, необхідні для виконання даного завдання. Мета - перейти від філософії WYSIWYG (What You See Is What Ооо Get - що бачиш на екрані, то і отримаєш в результаті) до філософії WYSIWYN (What You See Is What You Need - на екрані бачиш те, що тобі потрібно). Інтерфейси WYSIWYN залишають видимими ті, і тільки ті елементи, які дійсно потрібні користувачеві для виконання операції. З одного боку, в завдання проектування входить створення такого інтерфейсу, на якому були б явно видно всі потрібні і важливі функції. З іншого боку, хороший інтерфейс не повинен завалювати користувача занадто великою кількістю можливих варіантів або бентежити його надлишковою інформацією. WYSIWYN-інтерфейси краще вже тим, що вони беруть до обмеженість обсягу «оперативної пам'яті» людини і здатність впізнавати речі швидше, ніж згадувати. Навантаження на довготривалу пам'ять зменшується за рахунок того, що користувач постійно бачить всі необхідні опції і варіанти. На короткочасну пам'ять навантаження знижується за рахунок того, що користувачеві не доводиться запам'ятовувати і потім відтворювати інформацію, що міститься в якійсь іншій частині інтерфейсу.

**Принцип зворотного зв'язку**

Повідомляйте користувачам про дії системи, її реакціях, зміни стану або ситуації, про помилки і винятки, які важливі для них. Повідомлення повинні бути чіткими, короткими, однозначними і написаними на мові, зрозумілій користувачеві. Хороші інтерфейси знаходяться в діалозі з користувачами, повідомляючи їм про те, що відбувається в системі. Принцип зворотного зв'язку вказує розробникам деякі правила цього діалогу.

Практичні системи інформують користувача про безліч речей. Наприклад, вони повинні дозволяти йому дізнаватися про те, як сприймаються вводяться ним дані. Всякий раз, коли змінюється внутрішній стан системи, і це може зробити який-небудь вплив на роботу користувача, його слід повідомляти про це, особливо якщо змінюється інтерпретація системою його дій.

Зрозуміло, користувач повинен знати про дії, які заборонені або ігноруються. При цьому принцип зворотного зв'язку не може служити виправданням створення нескінченних віконець повідомлень. Інформування користувача - не самоціль, а спосіб організації діалогу в компактній і природній формі. Користувачам також потрібні повідомлення про помилки і виняткових ситуаціях. У багатьох програмах ці повідомлення, на жаль, неінформативні і здатні ввести в оману. Можна іноді зустріти навіть ображають повідомлення, після прочитання яких користувачеві може стати не по собі. Навряд чи людина надихнеться, скажімо, таким написом: «Неправильно! Введіть коректні дані! ». Таке повідомлення не тільки неявно передбачає, що користувач - якийсь нехороша людина, але і, по суті справи, не дає ніякої інформації. Тут не сказано, що саме неправильно і чому. Грамотно складені повідомлення про помилки - це ще один приклад хорошої організації спілкування з користувачем. Рекомендації тут можна дати такі: стислість; мову, зрозумілу користувачеві; простота розуміння. Перш за все інформативним повинен бути заголовок повідомлення. Він повинен в стислій формі описувати проблему, а вже саме повідомлення повинно розкривати подробиці і пропонувати шляхи вирішення або послідовність коригуючих дій.

**Принцип толерантності**

Інтерфейс повинен бути гнучким і толерантним. Збиток, нанесений помилками користувачі, необхідно знижувати за рахунок можливості скасування і затримки дій і за рахунок запобігання появ цих помилок шляхом аналізу різних форматів введення та розумної інтерпретації будь-яких

розумних дій. Інтерфейс можна робити більш-менш толерантним в залежності від того, які дані перевіряються і коли. Перевірка всіх полів разом з закінченням введення даних - практика поширена і іноді виправдана. При цьому толерантності системі додасть автоматичне підсвічування поля з неправильними даними, установка на нього курсора, плюс короткий, інформативне повідомлення в рядку стану. Найбільше користувачі страждають від програм, які після закінчення введення в усі поля перевіряють всю форму, і в разі неправильних даних хоча б в одному з полів користувач опиняється знову один на один з порожнім бланком. Здавалося б, таке рішення жахливо безглуздо, але воно зустрічається дуже часто, в тому числі і в комерційних програм-

мах. В цілому перевірка невикористовуваних полів або полів, які ніяк не обробляються системою і становлять інтерес тільки для користувачів (в тому вигляді, в якому вони були введені), веде до зниження толерантності ПО. Наприклад, перевірка того, що в поле приміток присутні тільки букви і цифри символи, надлишкова. Крім того, якщо користувач раптом захоче виділити що-небудь в цьому полі спеціальні символи або за допомогою псевдографіки, у нього виникнуть проблеми.

**Принцип повторного використання**

Слід багаторазово використовувати внутрішні і зовнішні компоненти і принципи поведінки системи, підтримуючи стійкість осмислено, а не просто за рахунок надмірності. Це сприяє зменшенню обсягу інформації, яку користувачам доводиться запам'ятовувати і про яку доводиться думати кожен раз заново.

Застосовуючи повторне використання зовнішніх і внутрішніх компонентів і рішень, що поширюються на всю систему, розробник може створити не тільки більш цілісний інтерфейс, але і більш дешевий продукт. Прагнення до однієї лише стійкості підвищує вартість розробки, так і в ряді випадків виявляється сізіфовою працею. Потрібно прагнути до стійкості в контексті вирішуваних системою завдань і області її використання, стійкість у жодному випадку не може бути самоціллю, інакше вона може виглядати дещо дурнувато і навіть приводити, як не дивно, до невдалих з точки зору несуперечності системам.

Багато прийняті стандарти і загальновизнані компоненти користувальницького інтерфейсу являють собою приклади невдалих спроб реалізації стійкості. Стандартні програмні платформи часом нав'язують розробника погано продумані і невдало спроектовані рішення. Самі звичайні діалогові вікна, скажімо, можуть забезпечувати цілісність і при цьому бути дуже низькосортними, вибір стандартних гарячих клавіш виявляється випадковим і ніяк не відповідає принципу стійкості.

**Прототип інтерфейсу необхідно робити з використанням Microsoft Visio або у спорідненому програмному забезпеченні.**

