**Лекція 19. Технології конструювання програмного забезпечення. Прототипування програмних систем та** **розробка користувальницького інтерфейсу**

**На період** карантину в дистанційній формі навчання на надані в кінці запитання потрібно надати письмові відповіді, виклавши на платформу коледжу (<https://github.com/omc-college/ipz41-software_design>) або надіславши їх на електронну адресу викладача. Файл надавати з іменем у форматі:

**KPZ<Номер групи><Номер лекції / лабораторної>[-<Номер завдання>][літера позначення типу роботи L – лекція, R – лабораторна]<Прізвищеанглійською>**.

Наприклад, **KPZ4118L**buts.doc.

Відповіді повинні бути не довгими і змістовними. Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності відповідей-"близнюків" відповідь буде зараховуватися першому за часом надсилання.

**Строк виконання цієї роботи 04.05.2020**

**Місце моделей в технології конструювання ПЗ**

***Модель ПЗ*** *–* ***це формалізований опис системи ПЗ на певному рівні абстракції*.** Кожна модель визначає конкретний аспект системи, використовує набір діаграм та документів заданого формату, а також відбиває точку зору та є об’єктом діяльності різних людей з конкретними інтересами, ролями або задачами. **Графічні (візуальні) моделі –** це засоби для візуалізації, опису, проектування та документування архітектури системи. **Моделі є** основою взаємодії учасників проекту і гарантують коректність архітектури. Оскільки складність систем зростає, важливо мати ефективні методи моделювання та притримуватися стандарту обраної мови моделювання.

***Склад моделей*,** використовуваних в кожному конкретному проекті, та ступінь їх деталізації в загальному випадку залежать ***від наступних факторів*:**

- складності проектованої системи;

- необхідності повноти її опису;

- знань та навичок учасників проекту;

- часу, відведеного на проектування.

***Візуальне моделювання*** сильно вплинуло на розвиток технології конструювання програмного забезпечення (ТКПЗ) взагалі та CASE-засобів зокрема.

***CASE-технологія*** – це сукупність методів проектування ПЗ, а також набір інструментальних засобів, які дозволяють наочно моделювати предметну галузь, аналізувати цю модель на всіх стадіях розроблення і супроводу ПЗ, а також розробляти додатки відповідно до інформаційних потреб користувача на основі аналізу специфікацій.

***CASE*** (англ. *Computer-Aided Software Engineering*) — набір інструментів і методів програмної інженерії для проектування програмного забезпечення, що допомагає забезпечити високу якість програм, відсутність помилок і простоту в обслуговуванні програмних продуктів

***При структурному аналізі та проектуванні використовуються різні моделі, які описують:***

- функціональну структуру системи;

- послідовність виконуваних дій;

- передачу інформації між функціональними процесами;

- відношення між даними.

***Найбільш поширеними моделями перших трьох груп є:***

- функціональна модель SADT (Structured Analysis and Design Technique);

- модель IDEF3;

- діаграми потоків даних DFD (Data Flow Diagrams).

**Метод SADT** – це сукупність правил та процедур, призначених для побудови функціональної моделі об’єкту будь-якої предметної галузі. Функціональна модель SADT відображає функціональну структуру об’єкту, тобто дії, які він виконує, та зв’язки між цими діями. Моделі SADT (IDEF0) традиційно використовуються для моделювання організаційних систем (бізнес-процесів). Слід зазначити, що метод SADT успішно працює лише при описі добре специфікованих та стандартизованих бізнес-процесів у корпораціях, тому прийнятий у США в якості типового.

***Перевагами застосування моделей SADT* є**: повнота опису бізнес-процесу, жорсткі вимоги методу (як результат – моделі стандартного вигляду), відповідність підходу до опису процесів стандартам ISO 9000. Оскільки на Україні бізнес-процеси почали формуватися і розвиватись порівняно недавно, вони слабко типізовані, тому розумніше орієнтуватись на менш жорсткі моделі.

***Метод моделювання IDEF3*** призначений для таких моделей процесів, в яких важливо зрозуміти послідовність виконання дій та взаємозалежності між ними. Основою моделі IDEF3 є так званий сценарій процесу, який виділяє послідовність дій та підпроцесів аналізованої системи.

**Діаграми потоків даних (DFD)** – це ієрархія функціональних процесів, зв’язаних потоками даних. Мета такого представлення – продемонструвати, як кожний процес перетворює свої вхідні дані у вихідні, а також виявити залежності між цими процесами. Модель системи визначається як ієрархія діаграм потоків даних, які описують асинхронний процес перетворення інформації від її ведення у систему до видачі результатів споживачу. ***Діаграми потоків даних*** спочатку створювались як засіб проектування інформаційних систем (SADT – засіб моделювання систем взагалі) і мають більший набір елементів, які адекватно відображають специфіку таких систем.

Розглянуті моделі приблизно однакові з точки зору можливостей зображувальних засобів моделювання. Основним критерієм вибору того чи іншого методу є ступінь володіння ним з боку консультанта або аналітика, грамотність вираження своїх думок мовою моделювання.

Найбільш поширеним засобом моделювання даних (предметної галузі) є модель «сутність-зв’язок» (Entity-Relationship Model - ***ERМ***). Ця модель представляє собою підмножину об’єктної моделі предметної галузі.

Концептуальною основою об’єктно-орієнтованого аналізу та проектування (ООАП) ПЗ є об’єктна модель. Її основні принципи – абстрагування, інкапсуляція, модульність та ієрархія; основні поняття – об’єкт, клас, атрибут, операція, інтерфейс.

Більшість сучасних методів ООАП засновані на використанні уніфікованої **мови моделювання UML (Unified Modeling Language).**

Мову UML використовують всі крупні компанії-виробники ПЗ (Microsoft, Oracle, IBM, Hewlett-Packard, Sybase). Крім того, практично всі світові виробники CASE-засобів підтримують UML у своїх продуктах. Повний опис UML можна знайти на сайтах http://www.omg.org та http://www.rational.com.

***Стандарт UML містить наступний набір діаграм:***

**1) *структурні (structural) моделі*:**

- діаграми класів (class diagrams);

- діаграми компонентів (component diagrams);

- діаграми розташування (deployment diagrams);

**2) *поведінкові (behavioral) моделі:***

- діаграми варіантів використання (use case diagrams);

- діаграми взаємодії (interaction diagrams): діаграми послідовності (sequence diagrams) та кооперативні діаграми (collaboration diagrams);

- діаграми станів (statechart diagrams);

- діаграми діяльності (activity diagrams).

UML має механізми розширення, призначені для можливості адаптації мови моделювання розробниками до своїх конкретних потреб, не змінюючи при цьому його метамодель. ***Наявність механізмів розширення принципово відрізняє UML від таких засобів моделювання, як IDEF0, IDEF1X, IDEF3, DFD та ERM.*** Перераховані мови є сильно типізованими, оскільки вони не допускають довільної інтерпретації семантики елементів моделей. UML, допускаючи таку інтерпретацію, є слабко типізованою мовою.

***До його механізмів розширення належать*:**

- ***стереотипи*** – новий тип елементу моделі, який визначається на основі вже існуючого елементу; це механізм, який дозволяє розділяти класи на категорії;

- ***теговані*** (іменовані) значення – це пара рядків «тег=значення» або «ім’я=вміст», в яких зберігається додаткова інформація про будь-який елемент системи;

- ***обмеження*** – це семантичне обмеження, яке має вигляд текстового виразу природньою або формальною мовою, яке неможливо виразити мовою UML.

**Моделювання бізнес-процесів** є важливою складовою частиною проектів по створенню великомасштабних систем ПЗ. Відсутність таких моделей є однією з головних причин невдач багатьох проектів. Моделі бізнес-процесів є самостійним результатом з великим практичним значенням. Наразі у моделюванні бізнес-процесів переважає процес ний підхід. Його основний принцип полягає у структуруванні діяльності організації відповідно до її бізнес-процесів, а не організаційно-штатної структури. Процесний підхід може використовувати будь-які з вище перерахованих засобів моделювання. Однак наразі спостерігається тенденція інтеграції різноманітних методів моделювання та аналізу систем, яка проявляється у формі створення інтегрованих засобів моделювання.

Під час моделювання кожний аспект діяльності підприємства спочатку розглядається окремо, а після детального припрацювання всіх аспектів будується інтегрована модель, яка відбиває всі зв’язки між різними аспектами.

Моделювання бізнес-процесів передбачає побудову двох моделей: ***модель бізнес-процесів (Business Use Case Model) та модель бізнес-аналізу (Business Analysis Model).*** Для моделювання бізнес-процесів можуть бути використані діаграми мови UML, але при моделюванні діяльності великої компанії, яка не лише виробляє продукцію, але й надає послуги, необхідно застосовувати різні методики моделювання.

**Прототипування ПЗ**

***Прототипування (макетування) ПЗ* –** це етап розроблення ПЗ, процес створення прототипу програми – макету (чорнової, пробної версії) програми, як правило, з метою перевірки придатності пропонованих для застосування концепцій, архітектурних та/або інших технологічних рішень, а також для представлення програми замовнику на ранніх етапах процесу розроблення.

***Прототип дозволяє також одержати зворотній зв’язок від майбутніх користувачів, причому, саме тоді, коли це найбільш необхідно*:** на початку проекту, коли ще є можливість виправити помилки проектування практично без втрат. Прототипування спрямовано на перевірку концепції та мінімізацію ризиків на етапі розроблення програмного забезпечення.

**Цілі прототипування:**

**1) *перевірка концепції та моделювання процесів*** – прототип дозволяє максимально наблизити бачення майбутньої системи до реального функціонування, включаючи емуляцію робочих процесів з використанням тестових даних, а також оцінити зручність використання; робота з прототипом дозволяє своєчасно скоригувати вимоги до майбутнього ПЗ і передавати у розроблення лише перевірені та ретельно деталізовані завдання;

**2) *керування інвестиціями та мінімізація ризиків*** – верифікація концепції та деталізація вимог, яка досягається в процесі створення та оцінювання прототипу, дозволяє мінімізувати ризики при інвестуванні у розроблення ПЗ шляхом завчасного виявлення потенційних «вузьких» місць, точної пріоритезації задач та реалістичного планування бюджетів та термінів. Такий підхід забезпечує коректну реалізацію проектних вимог та ідей, закладених в концепцію майбутньої системи.

Головна мета створення прототипу – зняти невизначеність у вимогах замовника. Модель може приймати одну з трьох форм:

* + 1. паперовий макет або макет на основі ПК (зображає або накреслює людино- машинний діалог);
    2. працюючий макет (виконує деяку частину потрібних функцій);
    3. існуюча програма (характеристики якої потребують покращання).

На рис.1 показано. як прототипування засноване на багаторазовому повторюванні ітерацій, в яких приймають участь замовник та розробник.

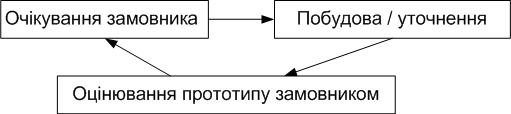


Рисунок 1. Прототипування

***Процес створення прототипу складається з таких кроків*:**

1) визначення початкових вимог;

2) розроблення першого варіанту прототипу, який містить лише інтерфейс користувача системи;

3) вивчення прототипу замовником та кінцевими користувачами, одержання зворотнього зв’язку про необхідні зміни та доповнення;

4) перероблення та покращення прототипу: з врахуванням одержаних зауважень та пропозицій змінюються як специфікації, так і прототип; після цього кроки 3 ф 4 можуть повторюватись.

**Два основних типи прототипування:**

**- *швидке прототипування*** (rapid або throwaway prototyping) – створюється макет, який на певному етапі буде залишений і не стане частиною готової системи. Основна перевага – у швидкості: у відповідь на свої вимоги замовник практично одразу одержує прототип інтерфейсу і одразу може уточнювати свої вимоги. Вартість зміни вимог на цьому етапі дуже низька, оскільки немає коду, який потрібно переписувати. Швидке прототипування виконується в найкоротші терміни, не обов’язково в межах тієї платформи та тих технологій, як і розроблювана система;

**- *еволюційне прототипування*** (evolutionary prototyping) – має за мету послідовно створювати макети системи, які будуть все ближче й ближче до реального продукту. Перевага такого підходу – на кожному кроці наявна робоча систем, яка нехай і не володіє всією необхідною функціональністю, але яка покращується з кожною ітерацією. При цьому немає витрат на код, який не буде використовуватись. Еволюційний підхід до прототипування обирається, якщо всі необхідні вимоги до моменту початку розроблення невідомі і будуть визначатись по мірі створення програми, тоді на кожному етапі реалізуються наявні та ясні вимоги. В деяких випадках, коли розробляється зовсім нова система, аналогів якої немає, користувачі починають використовувати систему ще до того, як вона буде повністю дописана, адже система з неповною функціональністю краще, ніж її повна відсутність.

**Переваги застосування прототипування:**

- **зменшення часу, вартості, ризиків:** прототипування покращує якість специфікацій; чим пізніше проводяться зміни у специфікації, тим вони дорожчі, тому уточнення «чого ж замовники хочуть насправді» на ранніх стадіях розроблення знижує загальну вартість;

- **залучення користувача у процес розроблення:** прототипування залучає майбутніх користувачів до процесу розроблення і дозволяє їм бачити те, як саме виглядатиме майбутня програма, що дозволяє позбавитись від можливих розбіжностей в уявленні про програму між розробниками та користувачами.

**Недоліки застосування прототипування:**

**- недостатній аналіз:** концентрація зусиль на обмеженому прототипі може відволікати розробників від необхідного аналізу вимог на повну систему;

- **змішування прототипу та готової системи в уявленні користувачів:** користувачі можуть подумати, що прототип, який пропонується відкинути і є основою майбутньої системи, тому можуть вимагати від прототипу більш точної поведінки або розчаруватись у можливостях розробників;

- **надмірний час на створення прототипу:** ключова властивість прототипу – короткий час його створення; якщо ж розробники не приймають це до уваги, то вони витрачають час на створення надто складного прототипу та втрачають переваги від застосування прототипування взагалі.

Спірним є питання, чи застосовне прототипування взагалі, у тій чи іншій формі, до всіх типів проектів. Однак відомо, що найбільші переваги прототипування дає при розробленні систем, які мають розвинутий інтерфейс користувача. Інакше прототипування майже не дає реальних переваг.

**Варіанти використання прототипів:**

- як інструмент видобування, перевірки та затвердження вимог;

- як основу для написання специфікації вимог до ПЗ та технічного завдання на етапі проектування;

- як техніку перевірки програмного дизайну на етапі проектування;

- як об’єкт дослідження юзабіліті-тестування (тестування на зручність використання);

- як зразок для розробників на етапі реалізації (конструювання);

- як зразок системи на етапі комерційної пропозиції;

- як зразок при тестуванні готового ПЗ;

- як зразок при прийманні-передачі роботи;

- як приклад рішення для демонстрації потенційним замовникам.

**Розробка користувальницького інтерфейсу.**

Прототипування є потужним інструментом при розробленні користувальницького інтерфейсу, оскільки дає користувачу наочно пересвідчитись в урахуванні його вимог ще до завершенні проекту.

Розробка користувальницького інтерфейсу є невід'ємною частиною будь-якого проекту пов'язаного зі створенням програмного забезпечення. Інтерфейс користувача є точкою взаємодії людини і програми, найчастіше має складну функціональність. Від того, наскільки зручним буде розроблений інтерфейс користувача, буде залежати і успіх продукту.

**Типи користувальницьких інтерфейсів**

На ранніх етапах розвитку обчислювальної техніки інтерфейс, призначений для користувача розглядався, як засіб спілкування людини з операційною системою і був досить примітивним. В основному він дозволяв запустити завдання на виконання, зв'язати з ним конкретні дані і виконати деякі процедури обслуговування обчислювальної техніки. Згодом, по мірі вдосконалення апаратних засобів, з'явилася можливість створення інтерактивного програмного забезпечення, що використовує спеціальні інтерфейси. В даний час основною проблемою є розробка інтерактивних інтерфейсів до складних програмних продуктів, розрахованих на використання непрофесійними користувачами. В останні роки були сформульовані основні концепції побудови таких користувальницьких інтерфейсів і запропоновано кілька методик їх створення.

***Інтерфейс*** являє собою сукупність програмних і апаратних засобів, що забезпечують взаємодію користувача з комп'ютером. Основу такої взаємодії складають діалоги.

***Під діалогом*** в даному випадку розуміють регламентований обмін інформацією між людиною і комп'ютером, який здійснюється в реальному масштабі часу і спрямований на спільне вирішення конкретного завдання: обмін інформацією та координація дій. Кожен діалог складається з окремих процесів введення-виведення інформації, які фізично забезпечують зв'язок користувача і комп'ютера.

Обмін інформацією здійснюється передачею повідомлень і керуючих сигналів. ***Повідомлення*** - порція інформації, яка бере участь в діалоговому обміні. розрізняють:

• вхідні повідомлення, які генеруються людиною за допомогою засобів введення: клавіатури, маніпуляторів і т. п.;

• вихідні повідомлення, які генеруються комп'ютером у вигляді текстів, звукових сигналів і/або зображень і виводяться на екран монітора для користувача або використовує для цього інші пристрої виведення інформації.

Об'єктно-орієнтовані інтерфейси використовують дещо іншу модель взаємодії з користувачем, орієнтовану на маніпулювання об'єктами предметної області. В рамках цієї моделі користувачеві надається можливість безпосередньо взаємодіяти з кожним об'єктом і ініціювати виконання операцій, в процесі яких взаємодіють кілька об'єктів.

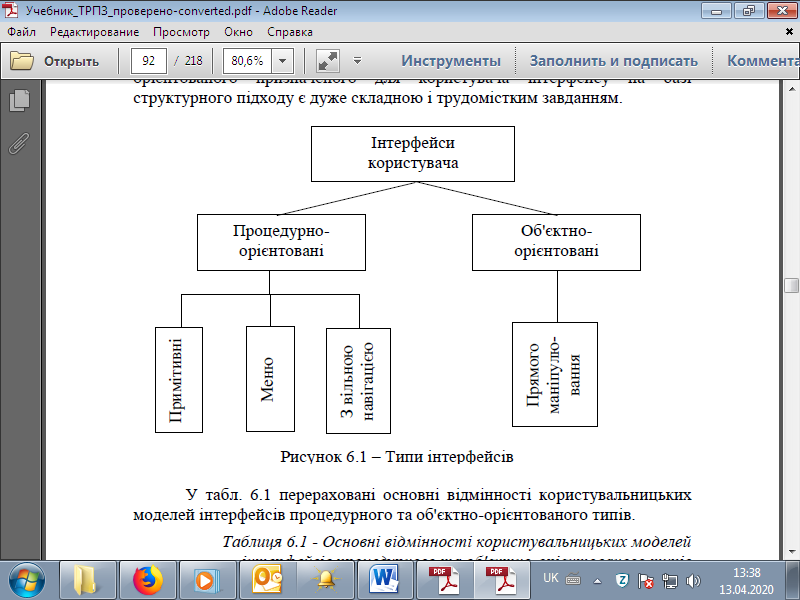


Рисунок 2. Типи користувальницьких інтерфейсів

Завдання користувача формулюється як цілеспрямована зміна деякого об'єкту, що має внутрішню структуру, певний зміст і зовнішнє символьне або графічне представлення. Об'єкт при цьому розуміється в широкому сенсі слова, наприклад, модель реальної системи або процесу, база даних, текст тощо. Користувачеві надається можливість створювати об'єкти, змінювати їх параметри і зв'язки з іншими об'єктами, а також ініціювати взаємодію цих об'єктів. Але реалізація сучасного процедурно- орієнтованого призначеного для користувача інтерфейсу на базі структурного підходу є дуже складним і трудомістким завданням.

У табл..1 перераховані основні відмінності користувальницьких моделей інтерфейсів процедурного та об'єктно-орієнтованого типів.

Таблиця 1. Основні відмінності користувальницьких моделей інтерфейсів процедурного та об'єктно-орієнтованого типів

| **Процедурно-орієнтований інтерфейс користувача** | **Об'єктно-орієнтований інтерфейс користувача** |
| --- | --- |
| *забезпечує* користувачів функціями, необхідними для виконання завдань | *забезпечує* користувачам можливість взаємодії з об'єктами |
| *акцент* робиться на задачі | *акцент* робиться на вхідні дані і результати |
| *піктограми* представляють додатки, вікна або операції | *піктограми* представляють об'єкти |
| зміст *папок і довідників* відображається за допомогою таблиць і списків | *папки і довідники* є візуальними контейнерами об'єктів |

**Типи процедурно-орієнтованих інтерфейсів**

Розрізняють процедурно-орієнтовані інтерфейси трьох типів: "примітивні", меню і з вільною навігацією.

***Примітивним*** називають інтерфейс, який організовує взаємодію з користувачем в командному режимі. Зазвичай такий інтерфейс реалізує конкретний сценарій роботи програмного забезпечення, наприклад: введення даних → рішення задачі → виведення результату.

Єдине відхилення від послідовного процесу, яке забезпечується даним інтерфейсом, полягає в організації циклу для обробки декількох наборів даних. Подібні інтерфейси в даний час використовують тільки в процесі навчання програмуванню або в тих випадках, коли вся програма реалізує одну функцію, наприклад, в деяких системних утилітах.

***Інтерфейс-меню*** на відміну від примітивного інтерфейсу дозволяє користувачеві вибирати необхідні операції зі спеціального списку, що виводиться йому програмою. Ці інтерфейси припускають реалізацію безлічі сценаріїв роботи, послідовність дій в яких визначається користувачем.

Розрізняють *однорівневі* і *ієрархічні* меню. Перші використовують для порівняно простого управління обчислювальним процесом, коли варіантів небагато (не більше 5-7), і вони включають операції одного типу, наприклад, *Створити*, *Відкрити*, *Закрити* і т. п. Другі - при великій кількості варіантів або їх очевидних відмінностей, наприклад, операції з файлами і операції з даними, що зберігаються в цих файлах.

Інтерфейси даного типу нескладно реалізувати в рамках структурного підходу до програмування. Алгоритм програми з багаторівневим меню зазвичай будується за рівнями, причому вибір команди на кожному рівні здійснюється так само, як для однорівневого меню.

Інтерфейс-меню передбачає, що програма знаходиться або в стані *Рівень меню*, або в стані *Виконання операції*. У стані *Рівень меню* здійснюється висновок меню відповідного рівня і вибір потрібного пункту меню, а в стані *Виконання операції* реалізується сценарій обраної операції. Як виняток іноді користувачеві надається можливість завершення операції незалежно від стадії виконання сценарію і/або програми, наприклад, після натискання клавіші Esc.

Деревоподібна організація меню передбачає суворо обмежену навігацію: або переходи "вгору" до кореня дерева, або - "вниз" за обраної гілки. Кожному рівню ієрархічного меню відповідає своє певне вікно, що містить пункти даного рівня. При цьому можливі два варіанти реалізації меню: кожне вікно меню займає весь екран або на екрані одночасно присутні кілька меню різних рівнів. У другому випадку вікна меню з'являються при виборі пунктів відповідного верхнього рівня - "випадаючи" меню.

В умовах обмеженої навігації незалежно від варіанту реалізації пошук необхідного пункту більше ніж дворівневого меню може виявитися непростим завданням.

Інтерфейси-меню в використовують в розробках, які повинні бути виконані по структурній технології і без використання спеціальних бібліотек.

***Інтерфейси з вільною навігацією*** також називають графічними інтерфейсами (GUI - Graphic User Interface) або інтерфейсами WYSIWYG (What You See Is What You Get – «що бачиш, те й отримаєш», тобто, що користувач бачить на екрані, то він і отримає при друці). Ці назви підкреслюють, що інтерфейси даного типу орієнтовані на використання екрану в графічному режимі з високою роздільною здатністю.

Графічні інтерфейси підтримують концепцію інтерактивної взаємодії з програмним забезпеченням, здійснюючи візуальний зворотний зв'язок з користувачем і можливість прямого маніпулювання об'єктами та інформацією на екрані. Крім того, інтерфейси даного типу підтримують концепцію сумісності програм, дозволяючи переміщувати між ними інформацію (технологія OLE).

На відміну від інтерфейсу-меню інтерфейс з вільною навігацією забезпечує можливість здійснення будь-яких дозволених в конкретному стані операцій, доступ до яких можливий через різні інтерфейсні компоненти, наприклад, меню різних типів: спадаюче, кнопкове, контекстне або різного роду компоненти введення даних. Причому вибір наступної операції в меню здійснюється як мишею, так і за допомогою клавіатури.

Суттєвою особливістю інтерфейсів даного типу є здатність змінюватися в процесі взаємодії з користувачем, пропонуючи вибір тільки тих операцій, які мають сенс в конкретній ситуації. Реалізують інтерфейси з вільною навігацією, використовуючи подієве програмування і об'єктно-орієнтовані бібліотеки, що передбачає застосування візуальних середовищ розробки програмного забезпечення.

***Об'єктно-орієнтовані інтерфейси з інтерфейсом прямого маніпулювання*** передбачає, що взаємодія користувача з програмним забезпеченням здійснюється за допомогою вибору і переміщення піктограм, відповідних об'єктів предметної області. Для реалізації таких інтерфейсів також використовують подієве програмування і об'єктно-орієнтовані бібліотеки.

Інтерфейс призначений для забезпечення взаємодії між користувачем і процесом, який виконує деяке завдання - прикладною програмою. Завданнями даної взаємодії є передача інформації (вихідних даних) від користувача прикладній програмі, вихідних даних (результатів роботи програми) користувачеві. Функцією інтерфейсу є також пояснення результатів роботи прикладної програми.

Орієнтація на кінцевого користувача означає, що інтерфейс повинен мати можливості для представлення вихідних даних і результатів у вигляді, загальноприйнятому в даній галузі, або в залежності від категорій користувачів і їх побажань: графічному, табличному, вербальному, причому кожне з них також може мати кілька видів уявлень.

**Фактори впливу на показники якості програмного продукту**

Дизайн користувальницького інтерфейсу є фактором, який впливає на три основні показники якості програмного продукту: його функціональність, естетику і продуктивність.

Функціональність є фактором, на який розробники додатків часто звертають основну увагу. Вони намагаються створювати програми так, щоб користувачі могли виконувати свої завдання і їм було зручно це робити. Функціональність важлива, але, тим не менш, це не єдиний показник, який повинен враховуватися в ході розробки.

Естетичний зовнішній вигляд самого додатка і способу його подання дозволяє сформувати у споживача позитивну думку про програму. Однак естетичні характеристики дуже суб'єктивні і описати їх кількісно більш важко, ніж функціональні вимоги або показники продуктивності. Вся естетика додатків часто зводиться до простого вибору: чи співвідносяться між собою використані кольори, чи передають елементи інтерфейсу їх призначення і сенс проведених операцій, що відчуває людина при використанні тих чи інших елементів управління і наскільки успішно їх використовує.

Продуктивність, а так само і надійність, також впливають на перспективу застосування програми. Якщо додаток добре виглядає, має просте і зручне управління, але, наприклад, повільно промальовує екрани, регулярно «підвисає» на десяток секунд, у нього, ймовірно, буде мало шансів на тривалу експлуатацію. У свою чергу, швидка і стабільна робота програми можуть частково компенсувати не самий стильний дизайн або відсутність якихось вторинних функцій.

**Етапи розробки призначеного для користувача інтерфейсу.**

Розробка користувальницького інтерфейсу включає ті ж основні етапи, що і розробка програмного забезпечення:

* збирання та аналіз вимог бізнес-замовника (постановка завдання) - визначення типу інтерфейсу і загальних вимог до нього; визначення специфікацій - розробка сценаріїв роботи користувачів;
* вибір показників оцінки ефективності призначеного для користувача інтерфейсу;
* проектування - проектування діалогів і їх реалізація у вигляді процесів введення-виведення (прототипування і розробка макетів);
* реалізація – програмування, розробка засобів підтримки користувачів і тестування інтерфейсних процесів і забезпечення якості інтерфейсу;
* розробка документації.

Ефективність роботи користувача визначається функціональними можливостями наявних в його розпорядженні апаратних і програмних засобів і доступністю для користувача цих можливостей.

*Для самостійного вивчення*: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

*Література* [1;2, 4, 12-14]

*Контрольні запитання для самоперевірки*.

1. З якою метою використовуються моделі в конструюванні ПЗ? .
2. В чому полягає моделювання бізнес-процесів?
3. Що таке прототипування та які його цілі??
4. Які типи прототипування Ви знаєте?
5. В чому переваги та недоліки прототипування?
6. Які типи користувальницького інтерфейсу Ви знаєте?
7. Що означає орієнтація інтерфейсу на кінцевого користувача?
8. Які фактори впливають на показники якості програмного продукту

*Контрольні запитання для надання письмових відповідей*.

1. На яких етапах ТКПЗ використовуються моделі і якого типу?
2. Які варіанти використання прототипів можна застосувати при розробленні користувальницького інтерфейсу?