1 ЛАБОРАТОРНА РОБОТА №1 "РОЗРОБКА ТЕХНІЧНИХ УМОВ"

Цілі роботи:

закріплення існуючих знань про методи і специфікації аналізу вимоги до інформаційних систем.

набуття навичок аналізу та формалізації вимог, представлений в ІР.

набуття навичок розробки технічних умов для створення Нова інформаційна система.

Завдання роботи:

1. Визначте вимоги до ІР.
2. Групові вимоги за групами: вимоги до системи в цілому; вимоги до функцій (завдань), що виконуються системою; вимоги до видів безпеки.
3. Пріоритетні вимоги (необхідні, бажані, додаткові)
4. Виберіть користувачів системи.
5. Виділіть основні випадки використання системи.
6. Запис сценаріїв для кожного вибраного випадку використання системи (з урахуванням вибраних вимог).
7. Презентуйте їх графічно за допомогою діаграм прецедентів і діаграм послідовностей.
8. Виконати аналіз постановки завдання по створенню ІП.
9. Визначити і сформулювати функціональні та технічні вимоги до інформаційної системи.
10. Розробити документ «Технічне завдання для створення ІП», що описує вимоги до ІП і містить іншу інформацію, необхідну для розробки.

Теоретична інформація

Вимоги до програмного забезпечення - це набір тверджень про атрибути, властивості або якості програмної системи, яка буде реалізована. Вимоги можуть бути виражені у вигляді текстових тверджень і графічних

моделей. Функціональні вимоги визначають необхідну поведінку програмної системи.

Існують наступні типи функціональних вимог:

1. Бізнес-вимоги – визначають призначення програмного забезпечення, описані в документі про зір і межі програмного проекту.
2. Вимоги користувача – визначити набір завдань користувача, які повинен вирішити програмний продукт, а також способи їх вирішення. Вимоги користувача можуть бути виражені у вигляді фраз заяв, випадків використання, сценаріїв взаємодії, історій користувачів.
3. Функціональні вимоги – охоплюють очікувану поведінку системи, визначаючи дії, які здатна виконати система. Процес розробки вимог включає в себе наступні кроки:

    1. визначення вимог (збір, розуміння, розгляд та уточнення потреб зацікавлених сторін);

    2. аналіз (перевірка цілісності та повноти); 3) специфікація (документація вимог);

3. перевірка правильності.

Розробка ТК включала в себе підготовку спеціального документа з такою ж назвою. Технічне завдання повинно описувати:

- обмеження, ризики, критичні фактори, що впливають на успішність проекту, наприклад, час відгуку системи на запит є заданим обмеженням, а не бажаним фактором;
- сукупність умов, за яких передбачається функціонування майбутньої системи: архітектура системи, апаратні та програмні ресурси, що надаються системі, зовнішні умови її функціонування, склад людей і роботи, що забезпечують безперебійне функціонування системи;
- терміни завершення окремих етапів, форма виконання робіт, ресурси, задіяні в розробці проекту, заходи щодо захисту інформації;
- описание выполняемых системой функций;
- будущие требования к системе в случае её развития, например возможность работы пользователя с системой с помощью Интернета и т.п.;

− сутності, необхідні для виконання функцій системи;
− інтерфейси та розподіл функцій між людиною та системою;
− вимоги до програмних та інформаційних компонентів програмного забезпечення, вимоги до СКБД. Якщо проект передбачається реалізувати для декількох СКБД, то вимоги до кожної з них, або загальні вимоги до абстрактної (наприклад, розподіленої) СКБД і перелік рекомендованих СКБД для цього проекту, що відповідають зазначеним умовам;
− що не буде реалізовано в рамках проекту.

Розробка ТК здійснюється відповідно до стандартів:

− ГОСТ 34.601-90. Інформаційні технології. Набір стандартів для автоматизованих систем. Автоматизовані системи. Етапи створення.
− ГОСТ 34.602-89. Набір стандартів для автоматизованих систем. Технічне завдання для створення автоматизованої системи.

І. Загальні положення

ТК повинна відповідати сучасному рівню розвитку науки і техніки, максимально точно відображати цілі, дизайн і вимоги до створюваної системи і в той же час не обмежувати розробника в пошуку і впровадженні найбільш ефективних технічних, технічних, економічних та інших рішень. Відповідно до ГОСТ 34.601-90, після узгодження з Замовником здійснюється розробка, проектування, затвердження та затвердження Технічного завдання АІС (при необхідності - з боку АІС). Цей стандарт також визначає склад учасників проектування та реалізації проектних рішень, які беруть участь у підготовці та (або) координації ТК. У найбільш загальному випадку до них відносяться:

1. Організація (користувач) замовника, для якої створюється АІС і яка забезпечує фінансування, прийняття робіт і експлуатації як по всій АІС, так і для окремих його компонентів;
1. Організація-розробник (генеральний проектувальник), що проводить роботи зі створення АІС, представляючи Замовнику комплекс науково-технічних послуг на різних етапах і етапах створення, а також розробку і постачання різних програмно-апаратних засобів ПС. Ця (материнська) організація може користуватися послугами інших організацій, що працюють на ній, на субпідряді;

2. Організація-постачальник, яка виробляє та (або) постачає програмне та апаратне забезпечення на замовлення Розробника або

Замовник;

3. Організації, що виконують будівельні, електричні, санітарні, монтажні, пусконалагоджувлі та інші підготовчі роботи, пов'язані зі створенням АІС.

ГОСТ 34.602-89 встановлює порядок розробки, затвердження та затвердження ТОР для створення (розробки або модернізації) автоматизованих систем різного призначення, а також склад і зміст цього документа незалежно від того, буде він працювати самостійно або в складі іншої системи. Будівля AIPS.

ТК на АІС розробляється на основі вихідних даних.

Будь-які зміни в TOR формалізуються додатковими протоколами, підписаними замовником і розробником. Таким чином, зроблені доповнення є невід'ємною частиною ТК на AIS. На титульній сторінці ТК повинен бути запис "Дійсний від ...".

СКЛАД І ЗМІСТ ТЕХНІЧНОГО ЗАВДАННЯ

Розглянемо склад ТК з урахуванням вимог ГОСТ 34.602-89.

ТК на AIS містить такі розділи:

1. Огляд.
2. Мета і цілі створення (розвитку) системи.
3. Характеристики об'єктів автоматизації.
4. Системні вимоги.
5. Склад і зміст роботи по створенню системи.
6. Порядок контролю та прийняття системи.
7. Вимоги до складу і змісту робіт з підготовки об'єкта автоматизації для введення В ЕКСПЛУАТАЦІЮ АІС в експлуатацію.
8. Вимоги до документації.
9. Джерела розвитку.
10. Застосування.

Залежно від типу, призначення, специфіки об'єкта автоматизації та умов функціонування системи дозволяється складати секції ТК у вигляді додатків, вводити додаткові, виключати або поєднувати підрозділи ТК.

Розглянемо зміст основних розділів ТОР з урахуванням вимог ГОСТ 34.602-89.

Розділ "Огляд":

1. Повна назва системи та її символ.
2. Найменування та реквізити підприємств (об'єднань) забудовника та замовника системи.
3. Перелік документів, які були підставою для створення системи, ким і коли вони були затверджені.
4. Можливі дати початку і завершення робіт зі створення системи.
5. Інформація про джерела та порядок фінансування робіт.
6. Порядок реєстрації та представлення замовнику результатів робіт зі створення системи або її частин, з виготовлення та коригування окремих засобів (технічних, програмних, інформаційних) та програмних систем системи.

Розділ "Мета та цілі створення (розвитку) системи":

1. "Призначення системи" означає тип автоматизованих процесів (діяльності) і перелік об'єктів, призначених для використання.
2. У пункті «Цілі системи» наведені назви і необхідні значення технічних, технологічних, промислово-економічних та інших показників об'єкта автоматизації, досягнутих в результаті створення АІС, вказуються критерії оцінки досягнення цілей системи.

Розділ "Характеристика об'єкта автоматизації":

1. Надає короткі відомості про об'єкт автоматизації або посилання на документи, які містять ці дані.
2. Інформація про умови експлуатації об'єкта автоматизації.
3. Характеристика зовнішнього середовища, в якому працює об'єкт автоматизації.

Розділ «Системні вимоги» містить підрозділи з вимогами до системи в цілому, функціями (завданнями), що виконуються системою, видами підтримки.

Вимоги до чисельності та кваліфікації персоналу АІС містять вимоги до кількості персоналу та користувачів АІС; кваліфікація персоналу, порядок його підготовки, контроль знань і навичок; режим роботи персоналу АІС.

Вимоги безпеки включають вимоги до забезпечення безпеки при монтажі, монтажі, експлуатації, обслуговуванні та ремонті технічних засобів системи (захист від впливу електричного струму, електромагнітних полів, акустичного шуму і т.д.), допустимі рівні освітленості, вібраційні і шумові навантаження.

Вимоги до збереження інформації містять перелік подій: аварії, збої технічних засобів (в тому числі втрати потужності) і т.д., в яких повинна бути забезпечена збереження інформації в системі, а також вимоги до підсистеми резервного копіювання і архівного зберігання документів і даних.

Вимоги до захисту інформації від несанкціонованого доступу включають вимоги, що діють в галузі (відділі) замовника.

Вимоги до ергономіки та технічної естетики включають показники АІС, які встановлюють необхідну якість взаємодії людини і машини і комфорт умов праці персоналу.

Вимоги до стандартизації та уніфікації включають показники, які встановлюють відповідність державним стандартам, відомчим та іншим нормам.

Додаткові вимоги можуть включати:

- вимоги до оснащення системи пристроями для навчання персоналу (тренажери, інші прилади аналогічного призначення) та документацією до них;
- вимоги до сервісних об'єктів, підставки для перевірки елементів системи;
- вимоги до системи, пов'язані зі спеціальними умовами експлуатації;
- спеціальні вимоги на розсуд розробника або замовника системи.

Підрозділ «Вимоги до видів підтримки», в залежності від типу системи, може містити вимоги до математичного, інформаційного, лінгвістичного, програмного, технічного, організаційного, методичного та інших видів системного забезпечення.

З точки зору вимог до математичної підтримки системи наведені вимоги до складу, сфери застосування (обмежень) і методів використання

математичних методів і моделей, типові алгоритми і алгоритми, що розробляються в системі.

З точки зору вимог до інформаційної підтримки системи видаються вимоги:

- до складу, структури та методів організації фондів та машинозчитуваних даних у системі;
- до обміну інформацією між компонентами системи;
- сумісність інформації з суміжними системами;
- про використання комунікативних форматів, уніфікованих документів, що діють в цій організації, та (або) взаємодіючої групи організацій;
- до внутрішньосистемних форматів даних;
- застосування систем управління базами даних;
- до структури процесу збору, обробки, передачі даних в системі і представлення даних;
- для захисту даних від руйнування в разі аварій і збоїв в електропостачанні системи;
- контролювати, зберігати, оновлювати та відновлювати дані;
- З точки зору вимог до лінгвістичної підтримки системи, вимоги до використання в системі наведені:
- класифікатори і тезауруси,
- мови взаємодії користувачів і технічні засоби системи,
- засоби кодування та декодування даних,
- перетворювачі,
- мови вводу/виводу даних,
- мови маніпулювання даними, –

способи організації діалогу.

З точки зору вимог до програмного забезпечення АІС наведені загальні функціональні та загальносистемні вимоги до придбаних та нещодавно розроблених програмних продуктів. Слід надати:

- вирішення за допомогою системного програмного забезпечення повного спектру сервісних і користувацьких завдань;

- підтримка можливості обробки, зберігання та оновлення зазначених видів документів і даних з урахуванням необхідних кількісних показників;
- підтримка можливості налаштування під задані вхідні та вихідні форми документів;
- Підтримка необхідних форматів даних та лінгвістична підтримка;
- підтримка вимог протоколів обміну телекомунікаційними даними, що діють у сфері функціонування АІС,
- забезпечення необхідної швидкості обробки та отримання даних для створеного АІС,
- забезпечення вимог стандартизації, уніфікації, ергономіки, інформаційної безпеки та дотримання інших вимог, не зазначених у цьому пункті, включених до інших пунктів ТОР.

З точки зору вимог до засобів технічного забезпечення системи призводять вимоги до видів технічних засобів, у тому числі до типів комплексів технічних засобів, програмно-технічних комплексів та інших компонентів, прийнятних для використання в системі, а також до функціональних, конструктивних і експлуатаційних характеристик засобів технічного забезпечення системи.

З точки зору вимог до організаційного забезпечення, вимог до структури та функцій підрозділів, задіяних у функціонуванні системи або забезпеченні роботи; організації функціонування системи та порядку взаємодії персоналу АІС з персоналом об'єкта автоматизації; захисту від помилкових дій персоналу Система.

Вимоги до управління та контролю включають:

- перелік контрольованих параметрів технологічного ланцюга обробки вхідних документів та обслуговування користувачів,
- вимоги до нормативних актів щодо обробки вхідних документів та обслуговування користувачів,
- вимоги до видів статистичної обробки контрольованих даних, а також до їх вихідних форм,
- вимоги до засобів формально-логічного контролю.

Розділ «Склад і зміст роботи зі створення (розробки) системи» повинен містити перелік етапів і етапів роботи зі створення системи відповідно до ГОСТ 34.601-90, терміни їх виконання, перелік організацій-виконавців робіт, посилання на документи, що підтверджують їх згоду на участь у створенні системи і т.д.

У розділі «Порядок контролю та прийому системи» зазначте:

1. Види, склад, обсяг і методи тестування системи та її компонентів (види випробувань відповідно до діючих стандартів, що застосовуються до розробленої системи);
2. Загальні вимоги до приймання робіт поетапно (перелік організацій-учасників, та/або юридичних та фізичних осіб, місце і терміни), порядок узгодження та затвердження приймальної документації;
3. Статус приймальної комісії (державної, міжвідомчої, відомчої тощо).

У розділі «Вимоги до складу і змісту робіт з підготовки об'єкта автоматизації до введення системи в експлуатацію» необхідно надати перелік основних заходів, які повинні виконуватися при підготовці об'єкта автоматизації до введення в експлуатацію АІС, і їх виконавців.

У розділі "Вимоги до документації" перелічено:

1. Перелік наборів і видів документів, що розробляються, в тому числі вироблених на верстатах, узгоджений розробником і замовником системи;
2. Вимоги до документації складових елементів міжгалузевого застосування відповідно до вимог ESKD і ESPD;
3. У разі відсутності державних стандартів, які визначають вимоги до документування елементів системи, додатково включають вимоги до складу і змісту таких документів.

Забезпечення якості проектної документації відноситься до здатності проектних інструментів аналізувати і перевіряти описи і документацію на повноту і узгодженість, а також на відповідність прийнятим стандартам і правилам (в тому числі ГОСТ, ESPD).

У розділі «Джерела розвитку» повинні бути перераховані документи та інформаційні матеріали (ТЕО, звіти про виконані науково-дослідні роботи, інформаційні матеріали про вітчизняних, зарубіжних системаханалогів і т.д.), на основі яких була розроблена ТК і які повинні використовуватися при створенні системи.

ТК на АІС включає додатки, що містять розрахунок очікуваної ефективності системи; оцінка науково-технічного рівня системи; використовуються при розробці ТК методичних і найважливіших інформаційних матеріалів з документів, зазначених в розділі «Джерела розвитку».

Додаткові рекомендації щодо складу і змісту ТК для автоматизованих систем різного призначення і додатків до них також містяться в РД 50-640-87 і ГОСТ 24.602-86.

ПРАВИЛА РЕЄСТРАЦІЇ ТК НА АІС

ТК оформляється на аркушах формату А4 без рамки, основного напису і додаткових стовпців до нього. Номери аркушів (сторінок) призначаються, починаючи з першого аркуша, що слідує за титульною сторінкою, у верхній частині аркуша (над текстом, посередині).

На титульній сторінці розміщені підписи замовника, розробника і координаційних організацій, які запечатані печаткою. При необхідності титульна сторінка оформляється на декількох сторінках. На останньому аркуші розміщені підписи розробників ТК на АІС та посадових осіб, задіяних у координації та розгляді проекту ТК на АІС.

При необхідності дозволяється розміщувати галузеві коди на титульній сторінці ТК, наприклад: робочий код, реєстраційний номер ТК і т.д.

Розділи та підрозділи ТОР повинні бути розміщені в порядку, встановленому ГОСТ 34.602-89.

Якщо конкретні значення показників, норм і вимог не можуть бути встановлені в процесі розробки ТК на АІС, він складає запис про порядок встановлення і узгодження цих показників, норм і вимог «Кінцева вимога (значення) зазначається в процесі ... і узгоджується протоколом з ... на етапі ...». При цьому ніяких змін до тексту ТК на АІС вносити не вносяться.

Титульна сторінка додатку до ТК на АІС складається аналогічно титульній сторінці Технічного завдання. Замість імені
"Технічне завдання" написати "Додаток 1 ... до ТК на АІС...".

На наступних аркушах будуть заважати доповнення до ТК на AIS підставою для зміни, змістом змін і посиланнями на документи, відповідно до яких вносяться ці зміни.

При представленні тексту додатку до ТК повинні бути вказані номери відповідних пунктів, підпунктів, таблиць основного ТК і застосовуватися слова "замінити", "доповнити", "видалити", "викладені в новій редакції".

Фактична практика проектування АІС передбачає наступні етапи (етапи) проектування:

1. Передпроектний огляд, у тому числі:

- короткий опис початкового стану об'єкта автоматизації та середовища, в якому він працює;
- зазначення основних цілей і перелік завдань автоматизації;
- опис розширеної організаційно-функціональної структури обраного варіанту (опцій) побудови створюваної системи;
- техніко-економічне обґрунтування системи;
- розширений опис та основні вимоги до засобів інформаційно-лінгвістичного забезпечення;
- перелік та загальні вимоги до програмного та апаратного забезпечення;
- перелік і збільшена характеристика етапів створення системи, термінів їх реалізації;
- первинна оцінка вартіть показників виконання робіт;

1. Технічне завдання для системи в цілому і (або) її основних компонентів (підсистем, програмно-апаратних комплексів і інструментів, індивідуальних завдань і т.д.), виконаних відповідно до ГОСТ 34.601-90.
2. Попередній дизайн. При розробці програмного забезпечення системи, Попередній дизайн повинен містити повну специфікацію програм, що розробляються.
3. Дослідно-промислова експлуатація розробленого АІС.

За результатами експериментальної, а часом і промислової експлуатації системи оптимізують роботу своїх компонентів і взаємодію між ними, в тому числі з урахуванням виконання співробітниками дій, визначених для них Технічним завданням і Проектом. Це не означає, що з часом цілі, завдання, способи їх досягнення, використовувані технічні та програмні засоби залишаються незмінними.

У процесі реального проектування складно реалізувати всі рекомендації, пов'язані з реалізацією найбільш ефективних

рішень. У зв'язку з цим оцінка отриманих результатів здійснюється шляхом порівняння основних показників з тими, що реалізуються в існуючих проектах. дозволяє досягти оптимальних рішень. Такі проекти називають квазіоптимальним (тобто кращими з раніше створених подібних проектів).

Порядок виконання лабораторних робіт

1. Ознайомтеся із запропонованим варіантом опису предметної області. Проаналізуйте предметний напрямок, уточнюючи і доповнюючи її, керуючись власним досвідом, консультаціями та іншими джерелами.
2. Виконати структурний поділ предметної області на окремі підрозділи (відділи, служби, підсистеми, групи і т.д.) відповідно до виконуваних ними функцій.
3. Визначити завдання і функції системи в цілому і функції кожної одиниці (підсистеми).
4. Виконати усний опис роботи кожної одиниці (підсистеми), алгоритми і сценарії їх індивідуальної роботи.
5. Вивчити вимоги до структури та змісту документа «Технічне завдання для створення ІП». Сплануйте документ.
6. Сформулювати цілі і завдання для створення ІП. Для характеристики типу ІП, його призначення, даних, що використовуються в роботі системи. Сформулюйте концептуальні вимоги до ІР.
7. Охарактеризувати типовий об'єкт автоматизації (організації, підприємства), для якого створюється ІП і на якому буде реалізований ІП. Опишіть автоматизовані бізнес-процеси.
8. Сформулюємо вимоги до системи в цілому. Опишіть структуру ІР. Список функціональних підсистем.
9. Сформулює функціональні вимоги. Опишіть вимоги до функцій і завдань, що виконуються системою. Опишіть призначення і склад функцій кожної з підсистем.

1. Описать предметную область. Разработать концептуальную модель данных предметной области. Сформулировать требования к информационному обеспечению системы.

1. Сформулює вимоги до системного програмного забезпечення. Опишіть вимоги до інтерфейсу користувача. Сформулюємо технічні вимоги до режимів реалізації та роботи ІП.
2. Використовуючи отримані результати, підготуйте документ «Технічне завдання для створення ІП», який включає в себе повний опис концептуальних, функціональних і технічних вимог до створеної системи.

Питання безпеки

1. Вимоги до інформаційної системи.
2. Методи аналізу і уточнення вимог.
3. Аналіз предметного простору.
4. Розробка технічних умов для створення інформаційної системи.
5. Концептуальні вимоги.
6. Функціональні вимоги.
7. Технічні вимоги.
8. Технології та методології управління вимогами.

2 ЛАБОРАТОРНІ РОБОТИ No2 «МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ»

Цілі роботи:

закріплення існуючих знань про технології та методології моделювання інформаційних систем; набуття навичок об'єктно-орієнтованого аналізу, Моделювання та дизайн IP; набуття навичок розробки моделі з використанням уніфікована мова моделювання UML.

Завдання роботи:

1. Розробити прецедентну модель, яка описує бізнес-процеси організації з точки зору зовнішнього користувача (клієнта) і відображає зовнішній погляд на діяльність організації. Результатом моделювання є діаграми активності та діаграми інцидентів.
2. Розробити модель бізнес-об'єктів, яка описує реалізацію бізнес-процесів організації її внутрішніми виконавцями. Основними складовими моделі є зовнішні і внутрішні виконавці. Результатом моделювання є послідовні діаграми.
3. Розробити концептуальну модель даних, яка описує об'єкти предметної області і відносини між ними. Результатом моделювання є діаграми класів і об'єктні діаграми.
4. Розробити опис системних вимог. Результатом є вичерпний перелік функцій, які повинні бути реалізовані в системі, і докладний опис необхідного виконання цих функцій.
5. Розробка моделей і додатків баз даних, які є докладним описом проекту бази даних і клієнтських додатків інформаційної системи. Результатом моделювання є складові діаграми і діаграми баз даних.

Теоретична інформація

Розробка інформаційної системи неможлива без її ретельного проектування: вплив цього кроку на наступні етапи життєвого циклу

інформаційної системи, яка базується на створеній базі даних, занадто великий.

Для проектування інформаційної системи можуть використовуватися наступні типи моделей:

1. методологія функціонального моделювання робіт САДТ (структурований аналіз і техніка проектування);
2. диаграммы потоков данных DFD (Data Flow Diagrams);
3. методологія проектування об'єктів в UML (UMLdiagrams).

Методологія SADT (Structured Analisys and Design Technique) була розроблена Дугласом Т. Россом і є одним з найвідоміших і широко використовуваних методів проектування. Нова назва методології, прийнятої в якості стандарту, -IDEF0 (Icam DEFinition) є частиною програми ICAM (Інтегроване автоматизоване виробництво).

Процес моделювання в SADT передбачає збір інформації про досліджувану область, документування отриманої інформації, представлення її як моделі та вдосконалення моделі. Крім того, цей процес передбачає чітко визначений спосіб виконання послідовного і надійного структурного розкладання, що є ключовим моментом у кваліфікованому аналізі системи.

У IDEF0 система представлена у вигляді набору взаємодіючих робіт (або функцій). Взаємозв'язки між робочими місцями визначають технологічний процес або структуру відносин всередині організації. Модель SADT являє собою ряд діаграм, які розщеплюють складний об'єкт на його складові частини.

Основними поняттями методології функціонального моделювання робіт є:

Роботи (діяльність) - іменовані процеси, функції або завдання, які відбуваються з плином часу і мають впізнавані результати. На схемі робота представлена прямокутниками.

Вхід - матеріал або інформація, яка використовується роботою для отримання результату (стрілка, що надходить в ліву сторону).

Управління - правила, стратегії, стандарти, які керують роботою (стрілка, що надходить у верхню грань). На відміну від вхідної інформації, елемент управління не підлягає зміні.

Вихід - матеріал або інформація, яка виробляється роботою (стрілка, що йде з правого боку). Кожна робота повинна мати хоча б одну стрілку виходу, так як робота без результату не має сенсу і не повинна моделюватися.

Механізм - ресурси, які виконують роботу (персонал, машини, пристрої - стрілка, що надходить в нижню грань).

Виклик - це взаємодія однієї робочої моделі з іншою (стрілка, що виходить з нижньої грані).

Існує п'ять типів робочих зв'язків в IDEF0.

Зв'язок вводу-виводу виникає, коли вихід вищої роботи відправляється на вхід наступної роботи.

Вихідний-контроль відноситься до ситуації, коли вихід вищої роботи спрямований на управління наступною роботою.

Зворотний зв'язок виводу-вводу виникає, коли вихід нижньої роботи спрямований на введення вищої роботи. Використовується для опису циклів.

Вихідно-контрольний зворотний зв'язок відноситься до ситуації, коли вихід нижчої роботи спрямований на управління вищою.

Вихідно-механізм зв'язку відбувається, коли вихід однієї роботи спрямований на механізм іншої і показує, що робота готує ресурси для іншої роботи.

З перерахованих блоків будуються схеми робіт, що описують принципи функціонування системи.

Схеми потоку даних (DFD) використовуються для опису руху документів та обробки інформації як доповнення до IDEF0. На відміну від IDEF0, де система розглядається як взаємопов'язана робота, стрілки в DFD показують лише те, як об'єкти (включаючи дані) переміщуються з одного завдання на інше. DFD відображає функціональні залежності значень, обчислених у системі, включаючи вхідні значення, вихідні значення та внутрішні сховища даних. DFD - це графік, який показує рух значень даних зі своїх джерел через процеси, які перетворюють їх на своїх споживачів в інших об'єктах.

Розширення DFD містить процеси, які трансформують дані, потоки даних, які передають дані, активні об'єкти, які виробляють і споживають дані, а також сховища даних, які пасивно зберігають дані.

Схема потоку даних містить:

1. Процеси, які трансформують дані
2. Потоки даних, які переносять дані
3. Активні об'єкти, які виробляють і споживають дані
4. Зберігаються дані, які пасивно зберігають дані.

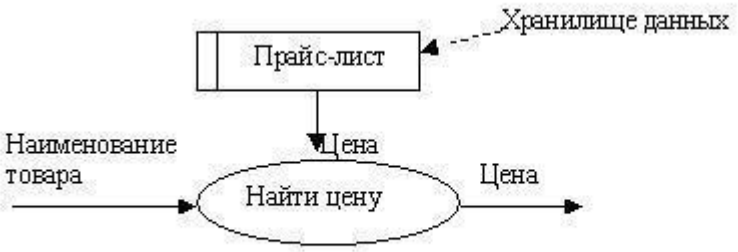
Процес DFD перетворює значення даних і представлений як еліпс, в якому розміщується ім'я процесу.

Потік даних з'єднує вихід об'єкта (або процесу) з входом іншого об'єкта (або процесу) і є проміжними обчислювальними даними. Потік даних представлений у вигляді стрілки між виробником і споживачем даних, позначених назвами відповідних даних. Дуги можуть розгалужуватися або зливатися, що означає, відповідно, ділення потоку даних на частини або злиття об'єктів.

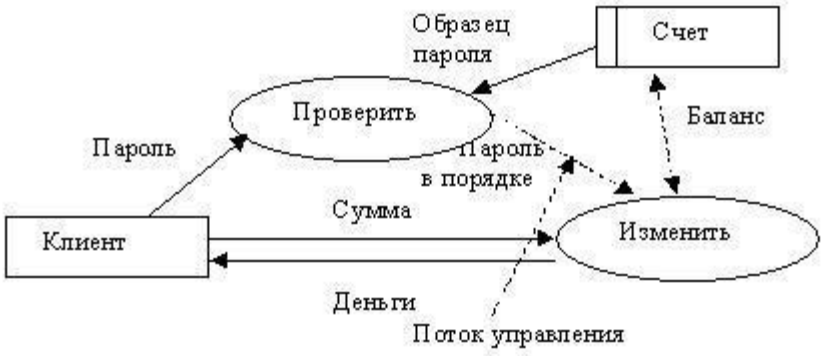
Активний об'єкт - це об'єкт, який забезпечує рух даних, забезпечуючи або споживаючи його. Сховище даних є пасивним об'єктом у межах DFD, в якому зберігаються дані для подальшого доступу.

Сховища даних. Сховище даних є пасивним об'єктом в межах DFD, в якому дані зберігаються для подальшого доступу. Сховище даних дозволяє

отримати доступ до даних, що зберігаються в ньому, в іншому порядку, ніж там, де він був розміщений. Сукупні сховища даних, такі як списки та таблиці, забезпечують доступ до даних у тому порядку, в якому він надходить, або за допомогою ключів.



Контроль потоків. DFD показує всі способи, якими обчислюються значення, але не показує, в якому порядку розраховуються значення. Рішення про порядок розрахунків пов'язані з програмним управлінням, що відбивається в динамічній моделі. Ці рішення, породжені спеціальними функціями або предикатами, визначають, чи буде виконуватися той чи інший процес, але не передають ніяких даних в процес, тому їх включення в функціональну модель не потрібно. іноді корисно включити ці предикати в функціональну модель так, щоб вона відображала умови виконання відповідного процесу.Функція, яка вирішує запустити процес, будучи включеною в DFD, генерує контрольний потік на схемі і представлена пунктирною стрілкою.



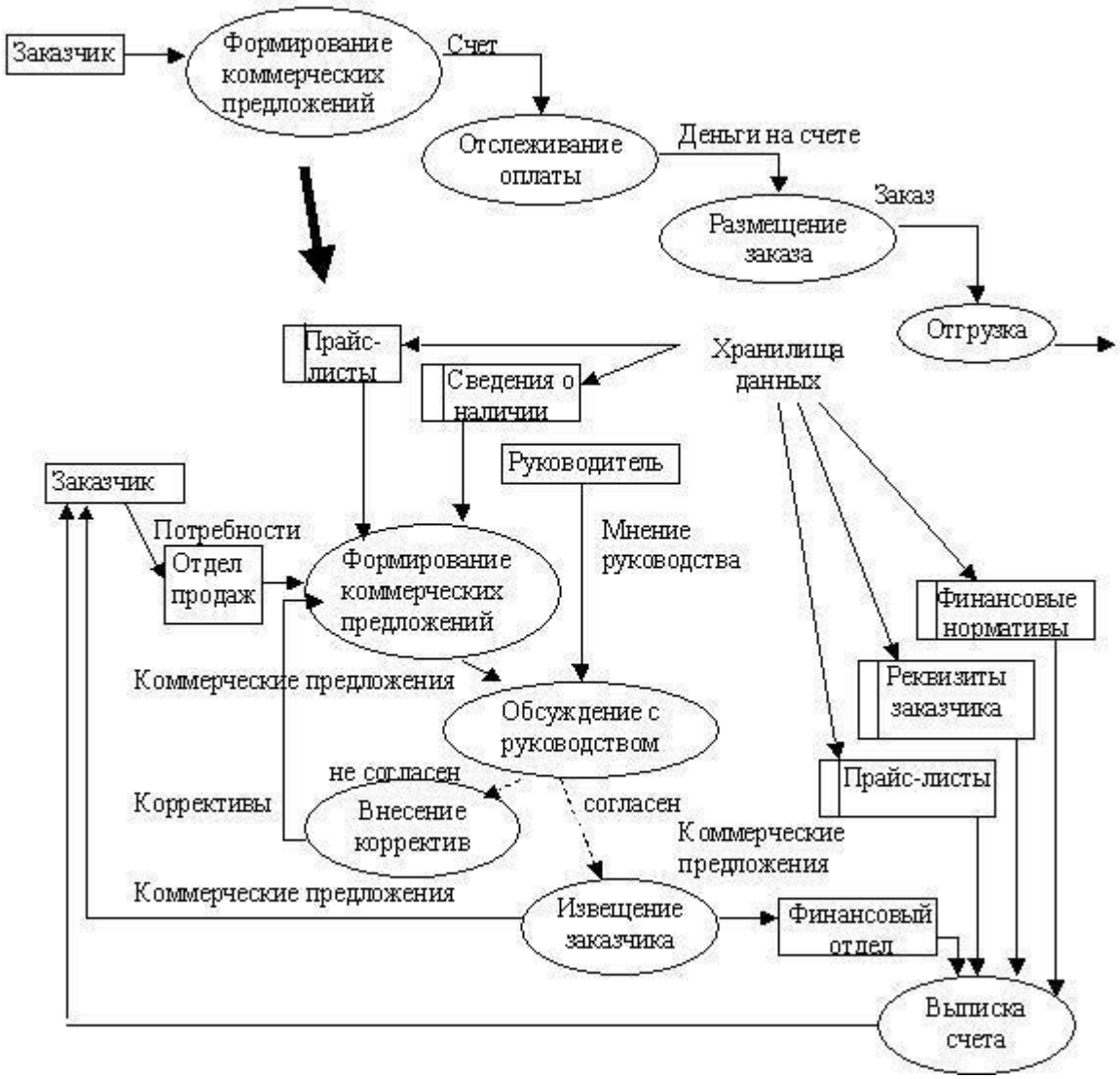
Першим кроком у побудові ієрархії DFD є створення контекстних діаграм. Зазвичай при проектуванні відносно простих інформаційних систем будується єдина контекстна діаграма з зіркоподібною топологією, в центрі якої знаходиться так званий основний процес, підключений до приймачів і джерел інформації, через які користувачі та інші зовнішні системи взаємодіють з системою.

Якщо для складної системи ви обмежитеся єдиною контекстною схемою, вона буде містити занадто багато джерел і приймачів інформації, які складно розмістити на аркуші паперу нормального формату, і, крім того, основний єдиний процес не розкриває структуру розподіленої системи.

Для складних інформаційних систем будується ієрархія контекстних діаграм. При цьому контекстна схема верхнього рівня містить не основний єдиний процес, а набір підсистем, з'єднаних потоками даних. Діаграми контексту наступного рівня деталізують контекст і структуру підсистем.

При побудові ієрархії DFD слід приступити до деталізації процесів тільки після визначення змісту всіх потоків і сховищ даних, який описується за допомогою структур даних. Структури даних побудовані з елементів даних і можуть містити альтернативи, умовні входження та ітерації. Умовний запис означає, що цей компонент може бути відсутній у структурі. Крім того, структура може включати в себе один з перерахованих елементів. Ітерація означає виникнення будь-якої кількості елементів у вказаному діапазоні. Для кожного елемента даних може бути вказаний його тип (неперервні або дискретні дані). Для неперервних даних може бути вказана одиниця виміру (кг, см і т.д.), діапазон значень, точність подачі і форма фізичного кодування. Для дискретних даних можна вказати таблицю припустимих значень.

Нижче наведено схему потоків даних верхнього рівня та їх подальше уточнення:



Універсальна мова моделювання (UML) - це мова графічного моделювання загального призначення, призначена для опису, візуалізації, проектування та документування всіх компонентів, створених у розробці програмного забезпечення. UML є об'єктно-орієнтованою мовою.

Його використання засноване на розумінні загальних принципів об'єктно-орієнтованого аналізу і проектування:

1. Принцип абстракції наказує включати в модель тільки ті аспекти розробленої системи, які безпосередньо пов'язані з виконанням функцій системи.
2. Принцип мультимоделінгу означає, що жодного єдиного представлення системи недостатньо, щоб адекватно виразити всі її особливості.

1. Принцип ієрархічної побудови моделей складних систем наказує розглядати процес побудови моделей на різних рівнях абстракції

або деталізації в рамках фіксованих уявлень. Діаграма UML - це графічне представлення набору елементів, зображених у вигляді зв'язаного графа з вершинами (сутностями) і ребрами (зв'язками), що використовуються для візуалізації системи з різних точок зору. Діаграми UML використовуються для опису різних аспектів функціонування та структури ІВ на різних етапах створення системи і, відповідно, на різних етапах моделювання: концептуальному, логічному та фізичному..

UML був розроблений rational software з метою створення найбільш оптимальної і універсальної мови для опису як предметної області, так і конкретного завдання в програмуванні. Візуальне моделювання в UML може бути представлено як процес спуску рівня від найбільш загальної і абстрактної концептуальної моделі системи до логічної, а потім і до фізичної моделі відповідної системи. Таким чином, будь-яка задача моделювалася за допомогою деякого набору ієрархічних діаграм, кожна з яких представляє деяку проекцію системи.

Діаграма є графічним представленням багатьох елементів. Найчастіше він зображується у вигляді з'єднаного графа з вершинами (сутностями) і ребрами (зв'язками). Існує вісім типів діаграм, визначених в UML:

1. Використовуйте діаграму випадку - схему поведінки, яка показує безліч прецедентів і акторів, а також взаємозв'язок між ними;
2. Діаграма діяльності - схема поведінки, яка показує машину і підкреслює переходи керуючого потоку від однієї діяльності до іншої;
3. Class diagram - структурна схема, яка показує безліч класів, інтерфейсів, кооперацій і взаємозв'язків між ними;
4. Діаграма statechart - схема поведінки, яка показує автомат і підкреслює поведінку об'єктів в плані порядку, в якому отримуються події;
5. Схема послідовності - схема поведінки, яка показує взаємодію і підкреслює часова послідовність подій;
6. Схема співпраці - схема поведінки, яка показує взаємодію і підкреслює структурну організацію об'єктів, які надсилають і отримують повідомлення;
7. Структурна схема - схема поведінки, яка показує автомат і підкреслює поведінку об'єктів з точки зору порядку, в якому події отримуються

8. Схема розгортання - структурна схема, яка показує вузли і зв'язки між ними.

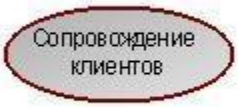
Прецедентна діаграма

Діаграми корпусів використовуються для моделювання вигляду системи з точки зору зовнішнього спостерігача. Діаграма прецедентів графічно показує набір прецедентів і суб'єктів, а також взаємозв'язок між ними.

Розглянемо основні елементи прецедентної діаграми.

Актор - **будь-яка сутність, що взаємодіє з системою ззовні [2] або сукупність логічно пов'язаних ролей, що виконуються при взаємодії з прецедентами [1].** Стандартним графічним позначенням предмета на схемах є фігура «людини», під якою записується конкретна назва предмета, але суб'єктом може бути не тільки людина, але і технічний пристрій, програма або будь-яка інша система, яка може служити джерелом впливу на змодельовані системи, як це визначається самим розробником.

Прикладом використання **є опис набору послідовностей дій (включаючи їх варіанти), які виконуються системою для того, щоб актор отримав для нього певне значення.** При цьому нічого не сказано про те, як буде здійснюватися взаємодія суб'єктів з системою, це одна з найважливіших особливостей розвитку прецедентів. Стандартним графічним позначенням прецеденту на схемах є еліпс, в рамках якого є коротка назва прецеденту або ім'я у вигляді дієслова з пояснювальними словами.



Суть концепції прецеденту має на увазі кілька важливих моментів.

1. Прецедент - це **повна** частина функціональності (включаючи основний потік логіки управління, будь-які її варіації (під течії) і виняткові умови (альтернативні потоки)).

2. Фрагмент **зовнішньо спостережуваних функцій** (крім внутрішніх функцій).
3. Ортогонаальна частина функціональності (прецеденти можуть ділитися об'єктами при виконанні, але кожен прецедент виконується незалежно від інших прецедентів).
4. Частина функціональності, **ініційована темою.** Після початку прецедент може взаємодіяти з іншими суб'єктами, і цілком можливо, що суб'єкт буде тільки в кінці прецеденту, опосередковано ініційованого іншим суб'єктом.
5. Частина функціональності, яка забезпечує **суб'єкту відчутний корисний результат**(і цей результат досягається в рамках єдиного прецеденту).

Можуть бути різні взаємозв'язки між суб'єктами і прецедентами, основні компоненти діаграми справ, які описують взаємодію екземплярів деяких суб'єктів і прецедентів з випадками інших суб'єктів і прецедентів. В UML існує кілька стандартних видів відносин між суб'єктами і прецедентами:

Відношення асоціації - *визначає існування* каналу **зв'язку між екземплярами суб'єкта і прецедентом (або між екземплярами двох суб'єктів).** На це вказує суцільна лінія, можна мати стрілку і вказувати на силу зв'язку.

Розширене співвідношення - **визначає взаємозв'язок між екземплярами єдиного прецеденту і більш загальним прецедентом, властивості якого визначаються на основі методу об'єднання цих екземплярів.** Він позначається пунктирною лінією зі стрілкою, що вказує на прецедент, який є розширенням до початкового прецеденту, і позначений ключовим словом "extend".

Включити співвідношення - **вказує на те, що якась дана поведінка для одного прецеденту включає в себе в якості складового компонента поведінку іншого прецеденту.** Це відношення є спрямованим бінарним відношенням в тому сенсі, що пара екземплярів прецедентів завжди впорядкована щодо включення. Позначається пунктирною лінією зі стрілкою, що вказує від базового прецеденту до включеного, і позначена ключовим словом "include" ("включає").

Узагальнення відношення - **служить для того, щоб вказати на те, що якийсь прецедент А можна узагальнити до прецеденту Б. У цьому випадку прецедент А буде спеціалізацією прецеденту Б.** При цьому Б називають предком або батьком по відношенню до А, а прецедент А є нащадком по відношенню до прецеденту Б. Слід підкреслити, що нащадок успадковує всі властивості і поведінку свого батька, а також може доповнюватися новими властивостями і поведінкою.

Приклад. Магазин відеопродукції.

Магазин продає відеокасети, DVD, аудіокасети, CDDisks і т.д., а також пропонує широкій публіці прокат відеокасет і DVD.

Товари поставляються декількома постачальниками. Кожна партія товару попередньо замовлена магазином у якогось постачальника і доставляється після оплати рахунку-фактури. Нещодавно отримані товари маркуються, вносяться в базу даних, а потім розподіляються на торговий майданчик або прокат.

Відео-носії орендуються на термін від 1 до 7 днів. При оренді з клієнта буде стягнуто заставу для відеоносця. При поверненні відеоносця повертається вартість застави мінус сума оренди. Якщо повернення коштів затримується менш ніж на 2 дні, стягується штраф у розмірі суми за оренду за 1 день* кількість днів прострочення. У разі затримки повернення більше 2 днів - сума вкладу не повертається. Клієнт може взяти одночасно до 4 відеоносців (прокатне замовлення). Квитанція видається для кожного відеоостерея.

Клієнти можуть стати членами відеоклубу і отримати пластикові картки. Членам клубу не стягується застава (крім випадку, описаного нижче), встановлюється знижка на орендну ставку і покупку товару. Учасники можуть попередньо замовити вибір відео для оренди або покупки.

У кожного члена клубу є певний ступок. Спочатку - "новачок". При поверненні 5 замовлень на оренду статус змінюється на «надійний». Якщо хоча б один відеооостерій затримується більш ніж на 2 дні, статус «початківець» або «надійний» змінюється на «ненадійний» і клієнту надсилається попередження. У разі повторного порушення правил статус змінюється на «порушника». Члени клубу зі статусом «надійних» можуть

приймати до 8 відеоносців одночасно, все інше - 4. З членів клубу зі статусом «порушника» береться застава.

Клієнти можуть оплатити готівкою або кредитною карткою при покупці товару або отриманні відеоносця в оренду.

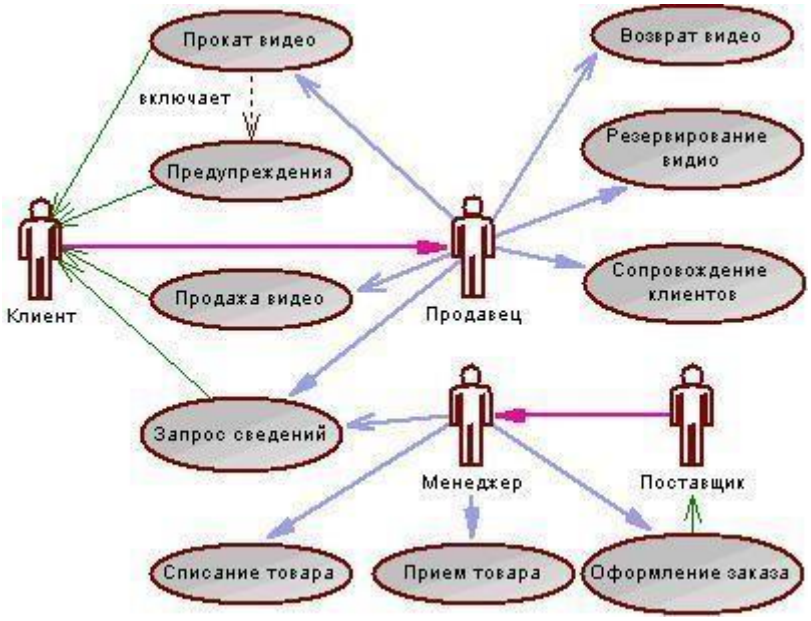
Прокат відеоносців після певної кількості днів оренди списуються і утилізуються актом. Також списуються товари та прокат відеоносців, які виявилися дефектними.

На малюнку нижче показано діаграму прецедентів для розглянутого прикладу. У цьому прикладі можна виділити наступні суб'єкти та відповідні їм прецеденти:

1. **Менеджер** - вивчає відеоринок, аналізує продажі (прецедент "Запит на інформацію"), працює з постачальниками: робить заявки на поставку товарів (прецедент "Замовлення"), оплачує і приймає товар (прецедент "Приймання товару"), списує товар (прецедент "Списання товару").
2. **Продавець** - працює з клієнтами: продає товар (прецедент "Продаж відео"), формалізує членство в клубі (прецедент "Підтримка клієнтів"), резерви (прецедент "Бронювання Відіо"), проблеми (прецедент "Оренда відео") і повертає відео-носії (прецедент "Повернення відео"), відповідає на питання клієнтів (прецедент "Запит інформації"). • **Постачальник** - оформляє документи для оплати товарів (прецедент "Замовлення"), доставляє товар (прецедент "Приймання товару"))
3. **Замовник** - купує (прецедент "Продаж відео"), орендує і повертає відео-носії (прецеденти "Відеопрокат" і "Повернення відео"), приєднується до клубу (прецедент "Підтримка клієнтів"), задає питання (прецедент "Запит інформації").

Останні два **суб'єкти господарювання, Постачальник і** Клієнт, не матимуть прямого доступу до розробленої системи (вторинних суб'єктів), але вони є основним джерелом подій, які ініціалізують прецеденти і одержувачами результату роботи прецедентів.

Від прецеденту «Оренда відео» до прецеденту «Попередження» коефіцієнт включення встановлюється на тій підставі, що кожен виданий відеоостереєм повинен бути перевірений на своєчасне повернення і, при необхідності, видається попередження клієнту.



Подальший розвиток моделі поведінки системи передбачає уточнення прецедентів. Для цього традиційно використовують два способи. Перший - це опис з використанням текстового документа. Такий документ описує, що повинна робити система, коли суб'єкт ініціював прецедент. Типовий опис містить такі розділи:

1. Підсумок
2. Актори-учасники
3. Передумови, необхідні для створення прецеденту
4. Потік подій (основний і, можливо, заглиборідний, альтернативний
5. Посткондицій, що визначають стан системи, після досягнення яких закінчується прецедент.

Діаграми активності, також звані діаграмами активності або діаграмами активності, були введені в мову UML відносно недавно. Діаграма активності - це, по суті, блок-схема, яка показує, як потік управління зміщується від однієї діяльності до іншої, з увагою, що фіксується на результатах діяльності. Результат може призвести до зміни стану системи або повернення певного значення.

Діаграма активності відрізняється від традиційної блок-схеми

1. більш високий рівень абстракції;
2. можливість представлення за допомогою схем управління паралельними потоками поряд з послідовним управлінням.

Основними використанням діаграм активності є

1. візуалізація особливостей здійснення класних операцій;
2. відображення внутрішньосистемної точки зору на прецедент.

В останньому випадку діаграми активності використовуються для опису кроків, які система повинна зробити після початку прецеденту.

Метою розробки схеми діяльності є:

3. деталізувати особливості алгоритмічної та логічної реалізації операцій і прецедентів, що виконуються системою;
4. виділяти послідовні і паралельні керуючі потоки;
5. підготувати детальну документацію для взаємодії розробників системи зі своїми клієнтами та дизайнерами.

Графічно діаграма діяльності представлена у вигляді графіка діяльності, вершинами якого є стани дії або стан діяльності, а дуги - переходи від одного стану дії / діяльності до іншого. Кожна діаграма діяльності повинна мати єдиний початковий і єдиний кінцевий стан (на практиці іноді можна побачити кілька скінченних станів на одній схемі, але це один і той же стан, зображений кілька разів для кращої читабельності діаграми). Зазвичай впорядковувати саму схему діяльності можна таким чином, щоб дії йшов зверху вниз. У цьому випадку початковий стан буде відображатися у верхній частині діаграми, а остаточний стан у нижній частині діаграми.

Розглянемо основні елементи діаграми діяльності.

Стан діяльності (діяльність, процес) - це **неатомний етап обчислень в автоматі, який триває в часі.** Стани активності можуть бути додатково розкладені, так що виконувана діяльність може бути представлена за допомогою інших діаграм діяльності. Стани діяльності не є атомними, тобто їх можна переривати. Передбачається, що для їх завершення потрібен помітний час.

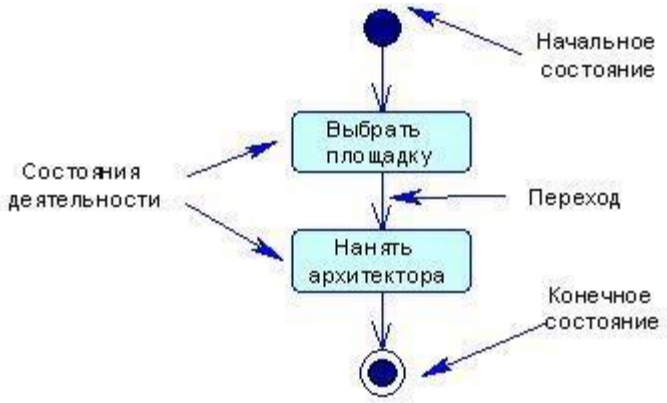
Стан дії- **стан, який представляє розрахунок атомної дії,** зазвичай заклик до операції. Стан дії не може бути розкладений. Вони атомні, тобто всередині них можуть відбуватися різні події, але робота, виконана в стані дії, не може бути перервана. операції, відправка сигналу, створення або знищення об'єкта, або в простому розрахунку - скажімо, значення виразу.

Стани діяльності і стан дії мають однакове стандартне графічне позначення - прямокутник з закругленими краями. Усередині такого

символу записується довільний вираз (action-вираз), який повинен бути унікальним в межах однієї схеми діяльності.

Початковий і кінцевий стани на діаграмах діяльності зображені у звиваються у зв'язків з малюнком і пофарбованим колом всередині кола відповідно.

Нижче наведено приклад діаграми діяльності:



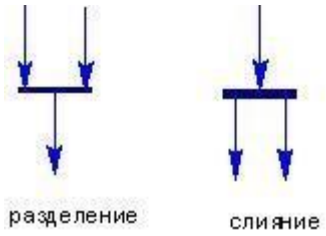
Переходи - зв'язок між двома станами, який вказує на те, що об'єкт в першому стані повинен виконати деякі дії і перейти до другого стану. Коли діяльність або діяльність в якомусь стані завершена, потік управління відразу переходить до наступного стану діяльності або діяльності. зі стрілкою.

Розгалуження. Прості послідовні переходи є найбільш поширеними, але тільки їх недостатньо для моделювання будь-якого потоку управління. Розгалуження описує різні шляхи виконання в залежності від значення деякого логічного виразу. Графічно точка розгалуження здається ромбом. Точка розгалуження може мати рівно один перехід і два або більше виходів. Для кожного вихідного переходу вказується логічний вираз, який обчислюється лише один раз при вході в точку розгалуження. Для відсутності двох вихідних переходів умови сторожового пса повинні бути одночасно встановлені на "true", в іншому випадку потік управління буде неоднозначним. Але ці умови повинні охоплювати всі можливі варіанти, інакше потік зупиниться.

Поділи та злиття. Прості і розгалужені послідовні переходи найчастіше використовуються в діаграмах діяльності. Однак часто виникає необхідність зобразити паралельні потоки, і це особливо актуально для моделювання бізнес-процесів. UML використовує панель синхронізації для

позначення поділу і злиття таких паралельних потоків виконання, яка малюється у вигляді жирної вертикальної або горизонтальної лінії. Поділ *(паралельна вилка) має одну вхідну вилку.* перехід і кілька вихідних, *що зливаються (одночасне з'єднання),*навпаки, має кілька вхідних переходів і один вихід.

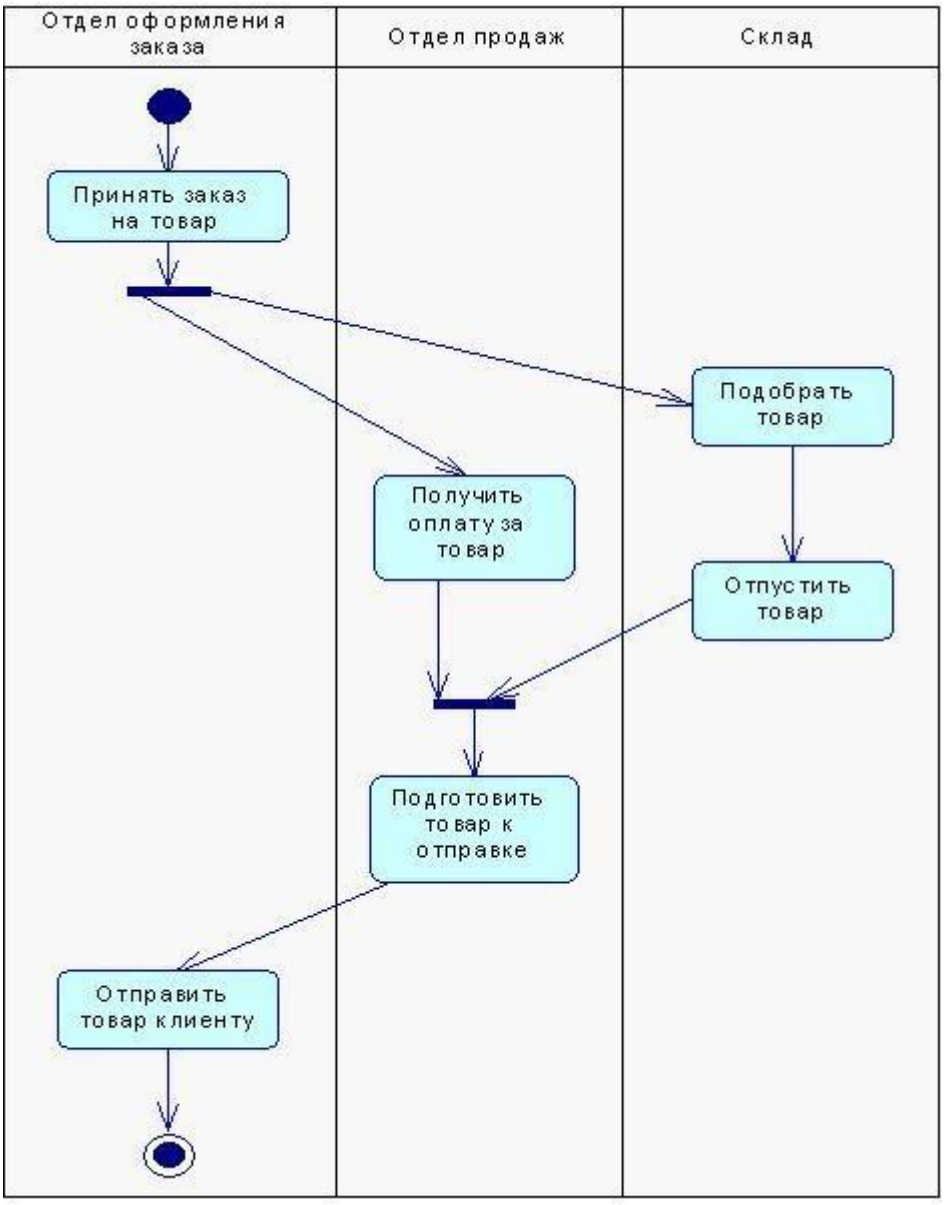
Приклад розщеплення і об'єднання потоків показаний нижче:



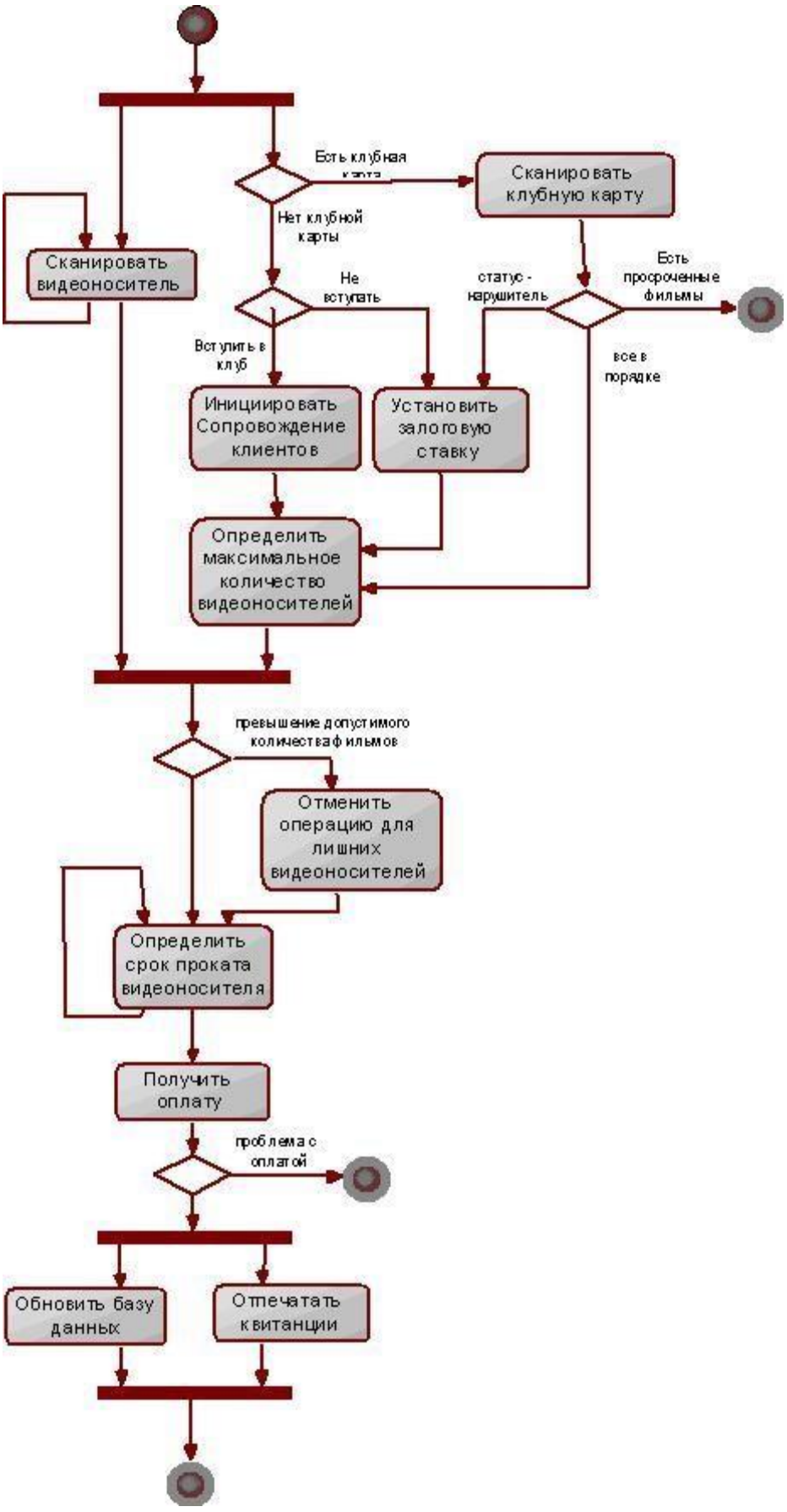
Трек. При моделюванню потоку бізнес-процесів іноді корисно розбити стан діяльності на діаграмах діяльності на групи, кожен з яких представляє відділ компанії, відповідальний за ту чи іншу роботу. У UML такі групи називаються шляхами (Swimlanes), так як візуально кожна група відділена від сусідніх вертикальною лінією, як доріжки для плавання в басейні (див. Рис. 3). Доріжки - це тип пакетів, які описують пов'язаний набір робіт.

Кожна композиція, присутня на діаграмі, отримує унікальне ім'я. Трек не несе ніякої глибокої семантики, за винятком того, що він може відображати якусь сутність реального світу. Кожна доріжка являє собою відповідальність за частину всієї роботи, показаної на схемі. На схемі діяльності, розбитій на доріжки, кожна діяльність належить рівно до однієї доріжки, але переходи можуть перетинати межі доріжок.

Існує певний зв'язок між доріжками і паралельними потоками виконання. Концептуально діяльність в рамках кожного треку, як правило, - але не завжди - розглядається окремо від діяльності в суміжних треках. Це розумно, тому що в реальному світі підрозділи організації, представлені треками, як правило, незалежні і функціонують паралельно.



На наступному малюнку показана діаграма прецеденту "Оренда відео" для прикладу "Відеомагазину", розглянутого в лабораторії 4



Порядок виконання лабораторних робіт

1. Ознайомитися з методикою структурного моделювання робіт.
2. Побудувати ряд робочих схем для всієї інформаційної системи в цілому і для окремих робочих сценаріїв, які відображають логіку і взаємозв'язок одиниць (підсистем).
3. Ознайомтеся з методологією схем потоку даних.
4. Створіть ряд схем потоку даних для окремих робочих сценаріїв, які відображають логіку і взаємозв'язок одиниць (підсистем).
5. Ознайомтеся з методологією моделювання прецедентів на основі мови UML.
6. Створіть діаграму інцидентів для області теми.
7. Опишіть кілька (2-3) прецедентів.
8. Ознайомтеся з методологією модельної діяльності на основі мови UML.

9 Створіть діаграми активності для кожного прецеденту, присутнього на діаграмі інциденту.

Контрольні питання

1. Моделювання інформаційних систем.
2. Типи моделей.
3. Структурні моделі IP.
4. Об'єктно-орієнтований аналіз і проектування.
5. Технології, мови та інструменти моделювання.
6. Уніфікована мова моделювання UML.
7. Діаграми UML: структурні діаграми, діаграми поведінки, діаграми взаємодії.
8. Інструменти моделювання IP.
9. Застосування UML в дизайні IP.

3 ЛАБОРАТОРНІ РОБОТИ №3 "ПРОЕКТУВАННЯ БАЗ ДАНИХ"

Мета роботи:

1. Об'єднання наявних знань про бази даних.
2. Вивчити методологію проектування баз даних як основи інформаційної системи.

Завдання роботи

1. Набувають навичок аналізу та формалізованого опису даної предметної області.
2. Набувають навичок розробки проекту бази даних з урахуванням його використання в рамках певної інформаційної системи.
3. Дізнайтеся, як побудувати інфологічні, даталогічні фізичні моделі, створені в процесі проектування баз даних:

Теоретична інформація

База даних - це сукупність даних, яка відображає стан об'єктів і їх взаємозв'язки в розглянутій предметній області. База даних є основою будь-якої інформаційної системи. Модель даних - це певна абстракція, яка при застосуванні до конкретних даних дозволяє користувачам і розробникам інтерпретувати їх як інформацію, тобто розглядати їх як інформацію, що містить не тільки дані, але і відносини між ними.

Реляційна модель даних заснована на концепції зв'язку, фізичне представлення якого являється двовимірною таблицею, що складається з рядків однієї і тієї ж структури. Логічна структура даних представлена набором пов'язаних таблиць. Система управління базами даних (СКБД) - це набір лінгвістичних і програмних засобів, необхідних для створення і використання бази даних. СКБД надає додаткам, розробникам і користувачам безліч різних уявлень про дані, що зберігаються в базі даних.

Інфологічна модель використовується на другому етапі проектування баз даних, тобто після словесного опису предметної області. Інфологічна модель - це опис майбутньої бази даних, представленої з

використанням природної мови, формул, графіків, діаграм, таблиць та інших інструментів, зрозумілих як розробникам баз даних (БД), так і звичайним користувачам. і чітка мова, що дає можливість залучати експертів з предмету, консультантів, користувачів для обговорення моделі і внесення виправлень. У цьому випадку під створенням інфологічної моделі буде розумітися її створення для БД. В цілому, інфологічна модель може бути створена для будь-якої розробленої системи і представляє її опис (в цілому, в довільній формі).

Створення інфологічної моделі є природним продовженням дослідження предметної області, але на відміну від неї є представленням бази даних з точки зору дизайнера (розробника). Видимість такої моделі дозволяє експертам домену оцінити її точність і внести корективи. Успіх подальшого розвитку залежить від правильності моделі.

Інфологічна модель може бути представлена у вигляді словесного опису, але найбільш показовим є використання спеціальних графічних нотацій, розроблених для такого роду моделювання.

Важливо відзначити, що створена на даному етапі модель повністю незалежна від фізичної реалізації майбутньої системи. У випадку з БД це означає, що не має значення, де будуть фізично зберігатися дані: на папері, в пам'яті комп'ютера і хто або що ці дані будуть обробляти. При цьому, коли структури даних не залежать від їх фізичної реалізації, а моделюються на основі їх семантичного призначення, моделювання називається семантичним.

Існує кілька способів опису інфологічної моделі, однак в даний час одним з найбільш широко використовуваних підходів, що використовуються в інфологічному моделюючому моделюючому, є підхід, заснований на використанні діаграм зв'язків сутностей (ER). При розгляді наступних прикладів ми будемо використовувати одну з найпоширеніших нот IDEF1X в моделях ER. Цей стандарт був розроблений в 1993 році Національним інститутом стандартизації і технологій і є федеральним стандартом обробки інформації (США), що описує семантику і синтаксис мови, правила і технології для розробки логічної моделі даних.

Для побудови інфологічної моделі важливо знати елементи цієї моделі. Основними елементами моделі зв'язку сутностей є сутності.

Суть за формою - це лише якийсь реальний опис об'єкта, а точніше набір описів його значущих особливостей. Певний набір значень атрибутів об'єкта буде називатися екземпляром сутності.

Даталогічна модель (DLM) побудована на основі бази даних ILM. DLM є концептуальною моделлю бази даних і відображає логічні зв'язки між інформаційними елементами DLM. У DLM записуються дані і зв'язки даних між ними.

DLM побудований з точки зору інформаційних одиниць, які діють в конкретних СКБД, в яких розроблена база даних. DLM залежить від вибору СКБД для розробки BND або інформаційної моделі. Схема бази даних - опис DLM мовою обраної СКБД.

Фізична модель використовується для прив'язки DLM до середовища зберігання фізичного шару. Ця модель бази даних визначається використовуваною пам'яттю, способами, якими дані фізично організовані в середовищі зберігання.

Фізична модель, як і DLM, побудована з урахуванням особливостей обраної СКБД.

Фізичний дизайн - це опис фізичної структури БД.

Розглянемо дизайн бази даних на прикладі моделі даних «Бібліотека».

Основні сутності включають:

1. Творці(Код Творця,Творець).

Ця організація призначена для зберігання інформації про основних людей, які брали участь у підготовці рукопису видання (автори, упорядники, редактори заголовків, перекладачі та художники). Таке злиття допустимо, так як дані про різних творців вибираються з одного домену (прізвище і імена) і виключає дублювання даних (одна і та ж людина може грати різну роль в підготовці різних публікацій). Наприклад, С.Ю. Маршак писав вірші (Повість про дурну мишу) і п'єси (Дванадцять місяців), перекладав Дж.

Оскільки прізвище і імена (ініціали) творця можуть бути досить громіздкими (М.Є. Салтиков-Щедрін, Франсуа Рене де Шатобріан, Остін Жюль Жан-Батист Іполіт і т.д.) і будуть неодноразово зустрічаються в різних виданнях, бажано їх пронумерувати і послатися на ці цифри. Для

цього введіть цілочисельний атрибут «Creator_Code», який буде автоматично збільшуватися на один при вході в базу даних нового автора, перекладача або іншого творця.

Аналогічно створені: Publisher_id, Title_code, Publication_type, Character_code, Language_code Ticket_number, Binding_number, Place_code і Edition_code, замінивши один-дев'ять атрибутів.

1. *Видавці (s_Code видавців,ім'я, місто).*
2. *Назви(Code_Headings,назва).*

Виділення цього суб'єкта господарювання зменшить обсяг даних і зменшить ймовірність невідповідності (позбавляє від необхідності введення довгих текстових назв для різних томів зібраних робіт, перевидань, підручників і т.д.).

3. *Type_of_edition (Type_of_edition,Name_of_view).*
4. *Персонажі (Character_Code,Reissue_Character).*
5. *Мови(Language_Code,мова, абревіатура).*

Крім назви мови зберігається її загальна абревіатура (англійська, іспанська, німецька, французька), якщо вона існує.

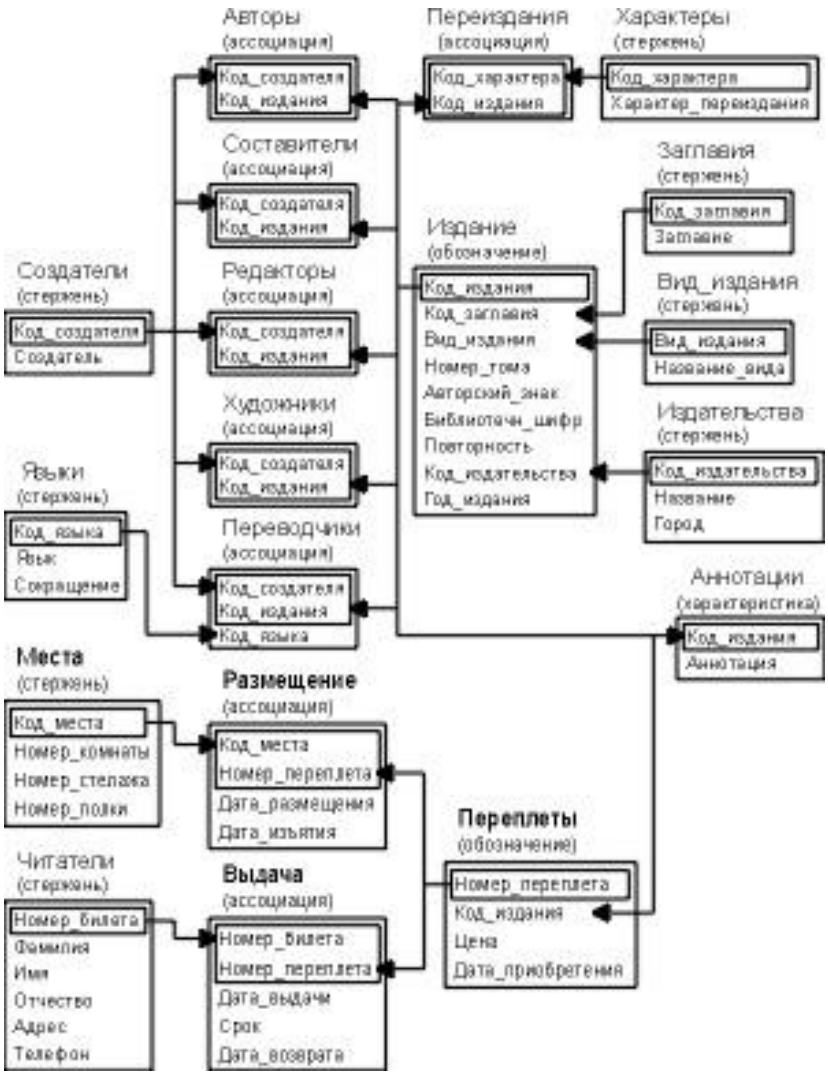
6. *Місця(Place_code,Room_number, Rack_number, Shelf_number).*

Один з кодів цієї сутності (наприклад, "-1") зарезервований для опису узагальненого місця, розташованого за стінами книжкового сховища (видання видається читачеві, тимчасово переноситься в іншу бібліотеку або організацію).

1. *Читачі (Ticket_number,прізвище, ім'я, по батькові, адреса, телефон).*

Дві ключові сутності, що описують видання та його конкретні копії, залежать від інших сутностей і потрапляють до класу нотацій:

1. *Видання (Publication_Code,Title_Code, Edition_View, Volume_Number, авторські s_mark, Library_cipher, Повторення, s_code, Year_of_publication, анотація видавця [Назви, Type_of_publication, Публікація];*
2. *Прив'язки (Binding_number, Publication_code, ціна, Date_of_purchase)[Видання];*



Інфологічна модель предметної області «Бібліотека»

Основні сутності та позначення пов'язані асоціаціями:

1. *Авторы [Создатели M, Издание N] (Код_создателя, Код_издания).*
2. *Составители [Создатели M, Издания N] (Код_создателя, Код_издания).*
3. *Редакторы [Создатели M, Издания N] (Код_создателя, Код_издания).*
4. *Художники [Создатели M, Издания N] (Код_создателя, Код_издания).*

5. *Переводчики [Создатели M, Издания N] (Код_создателя, Код_издания, Язык).*
6. *Переиздания [Характеры M, Издания N] (Код_характера, Код_издания).*
7. *Размещение [Места M, Переплеты N] (Код_места, Номер_переплета, Дата_размещения, Дата_изъятия).*
8. *Выдача [Читатели M, Переплеты N] (Номер_билета, Номер_переплета, Дата_выдачи, Срок, Дата_возврата).*

Тепер необхідно перевірити, чи порушуються в цьому проекті якісь принципи нормалізації, тобто що будь-яке не ключове поле кожної таблиці:

1. функціонально залежить від повного первинного ключа, а не від його боку (якщо ключ композитний);
2. не має функціональної залежності від іншого не-ключового поля.

Автори, упорядники, редактори, художники та Передруки, які не мають ключових полів, безумовно, нормалізуються. Нормалізовані та сутності Творці, Персонажі, Заголовки, Type_editions, що складаються з не складеного ключа та єдиного не-ключового поля.

Аналіз сутностей Перекладачі, Розміщення та Видачі, що складаються з складених ключових і не-ключових полів, показав, що між не-ключовими полями немає функціональних зв'язків. Останні функціонально не залежать від будь-якої частини композитного ключа.

Нарешті, аналіз сутностей публікації, прив'язки, місця, читачі та мови, показав, що єдиною «підозрілою» сутністю є ядро мов, яке має два функціонально пов'язаних не-ключових поля: мова та абревіатура.

Поле Мова не є ключовим, оскільки ви ввели language_code цифровий первинний ключ, який замінює можливий мовний ключ. Це дозволило зменшити обсяг даних, що зберігаються в таблиці Перекладачів, витрати на оплату праці при введенні різноманітних текстових значень і можливу невідповідність, яка часто виникає через введення помилкових дублікатів в різні поля (наприклад, "англійська", "англійська", "англійська", "англійська",

"англійська", "англійська" і т.д.).

Процес проектування бази даних, заснований на принципах нормалізації, являєеною переходів від неформального словесного опису інформаційної структури предметної області до формалізованого опису об'єктів предметної області з точки зору певної моделі.

Процес проектування тривалий і вимагає обговорення з замовником і з експертами з предмета. Нарешті, при розробці серйозних корпоративних інформаційних систем проект бази даних є фундаментом, на якому будується вся система в цілому, а питання можливого кредитування часто вирішується експертами банку на основі грамотно складеного проекту інформаційно-бази даних. Отже, інфологічна модель повинна включати формалізований опис предметної області, яку легко «прочитати» не тільки фахівці бази даних. І цей опис повинен бути настільки містким, що можна оцінити глибину і правильність вивчення проекту бази даних, і звичайно ж, він не повинен бути прив'язаний до конкретної СКБД. Вибір СКБД - це окреме завдання, для його правильного рішення необхідно мати проект, який не прив'язаний до якоїсь конкретної СКБД.

Порядок виконання лабораторних робіт

1. Складіть план розробки проекту бази даних для даної предметної області. База даних повинна розглядатися як частина майбутньої інформаційної системи, яка автоматизують бізнес-процеси якоїсь організації.
2. Виконати аналіз вказаної предметної області. Сформулюється словесний опис інформаційних об'єктів. Опишіть типові запити для пошуку та аналізу інформації про об'єкти області теми.
3. Побудувати інфологічну модель даних, яка описує предметну область в рамках ER-моделі «сутність - зв'язок». Отримайте візуальне представлення інфологічної моделі, побудувавши ERdiagrams.

1. Построить даталогическую модель базы данных. Преобразовать полученные ранее ER-модели в конкретную схему реляционной базы данных.

1. Перевірте повноту і правильність моделі даних бази даних шляхом складання стандартних запитів для пошуку та аналізу інформації.
2. Створення таблиць баз даних на основі моделі даних.
3. Визначення меж цілісності бази даних.
4. Створіть схему даних.

9.Розробити типові запити і переконайтеся, що база даних створена правильно.

Питання безпеки

1. Бази даних як основа інформаційної системи.
2. Види і призначення баз даних.
3. Моделі даних.
4. Системи управління базами даних.
5. Реляційні бази даних і СКБД.
6. Технології проектування баз даних.
7. Як можна уявити собі інфологічну модель?
8. Що таке сутності?
9. Як представлені базові таблиці?
10. Як будуються даніологічні та фізичні моделі?
11. Опишіть принципи нормалізації.

4 ЛАБОРАТОРНІ РОБОТИ No4 "ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА"

Мета роботи:зміцнити існуючі знання про дизайн інтерфейсу користувача.

Завдання роботи:

1. Набути навичок практичного застосування методів збору вимог до користувальницького інтерфейсу ІП.
2. Сформувати уявлення про зміст аналізу вимог до інтерфейсу користувача.
3. Отримайте навички дизайну інтерфейсу користувача та вибір дизайну.

Теоретична інформація

Користувальницький інтерфейс являє собою сукупність логічних і фізичних принципів взаємодії між користувачем і компонентами інформаційної системи

У дизайні інтерфейсу користувача умовно можна виділити два *компоненти: декоративний; активний.*

Декоративна складова включає елементи, відповідальні за *естетичну* привабливість програмного продукту.

Активні елементи інтерфейсу користувача поділяються на:

оперативний – для відображення результатів операцій над даними; *інформація* – для відображення інформації, що супроводжує виконання операцій: роз'яснень, коментарів, порад, повідомлень тощо; *менеджери* – для управління програмою.

Важливим принципом побудови дизайну інтерфейсу є баланс між активними і декоративними компонентами. Оскільки при створенні ігор головне - баланс між складністю гри і її захопленням, то інтерфейс повинен

бути забезпечений балансом між функціональністю програми, можливостями маніпулювання нею і її живописною серією.

Символічний інтерфейс

Символічний інтерфейс заснований на використанні різних символів (букв, цифр, знаків) для відображення інформації на екрані монітора і введення користувачем даних і команд для виконання.

Існує два типи інтерфейсів символів:

командний рядок; Символічні форми.

Командний рядок взаємодіє з користувачем з комп'ютером, надаючи комп'ютеру команди, які він виконує, і дає користувачеві результат їх виконання.

За допомогою цієї технології клавіатура служить способом введення оператором інформації в комп'ютер, а комп'ютер відображає інформацію людині за допомогою буквено-цифрового дисплея (монітора). Комбінація монітора-клавіатури стала відома як термінал або консоль. Команди вводяться в командному рядку, який є символом запрошення і миготливим курсором, а введені символи можуть бути стерті і відредаговані. Натиснувши клавішу «Enter», комп'ютер отримує команду і починає її виконувати. Після переходу на початок наступного рядка комп'ютер відображає результати своєї роботи на моніторі. Найбільш поширений командний інтерфейс був в операційній системі MS DOS.

Форми символів формуються на екранних ділянках монітора для введення інформації (форми) з використанням тільки символів. Введення даних також здійснюється виключно за допомогою символів. Для заповнення введення форми і початку її обробки комп'ютером даються команди за допомогою клавіш клавіатури.

Графічний інтерфейс

Графічний інтерфейс використовує растрові та векторні технології для побудови зображень на екрані монітора.

Растрова технологія генерує зображення, розфарбовуючи пікселі екрану в різні кольори.

Векторна технологія будує зображення на основі найпростіших геометричних фігур (відрізків, кривих, кіл і т.д.) Існує кілька типів графічних інтерфейсів:

WIMP-інтерфейс; тривимірні

інтерфейси.

В даний час широко використовується графічний WIMP-інтерфейс.

WIMP *("Window,* ***i****con,* ***M****enu,* ***P****ointer")* - це інтерфейс взаємодії людськоїмашини на основі елементів: "вікно, іконка, меню, покажчик"). Ця технологія була запропонована Merzoog Wilberts в 1980 році. Хоча його популярність поступово знижується, це слово часто використовується як близький синонім «графічного інтерфейсу користувача».

WIMP був розроблений **корпорацією Xerox PARC** і популяризований комп'ютером Macintosh в 1984 році (до нього додали концепцію «рядка меню» і концепцію розширеного управління вікнами).

Характерною особливістю інтерфейсу WIMP є те, що діалог користувача з комп'ютером ведеться не за допомогою командного рядка, а за допомогою вікон, графічних зображень меню, курсору та інших елементів. Хоча цей інтерфейс дає команди машині, але це робиться за допомогою графічних зображень.

Ідея графічного інтерфейсу виникла в середині 70-х років в дослідницькому центрі компанії Xerox Palo Alto Research Center (PARC). Обов'язковою умовою графічного інтерфейсу стало зменшення часу реакції комп'ютера на команду, збільшення обсягу оперативної пам'яті, а також розробка елементної бази, технічних характеристик комп'ютера і, зокрема, моніторів. Після появи графічних дисплеїв з можливістю виведення будь-яких графічних зображень різних кольорів графічний інтерфейс став невід'ємною частиною всіх комп'ютерів. Поступово відбувався процес уніфікації у використанні клавіатури і миші за допомогою прикладних програм. Злиття цих двох тенденцій призвело до створення користувальницького інтерфейсу, за допомогою якого при мінімальних часах і грошах на перепідготовку персоналу можна працювати з будь-якими програмними додатками Даний тип інтерфейсу реалізований у вигляді двох рівнів:

простий графічний інтерфейс; Full WINP

— інтерфейс.

*Простий графічний*інтерфейс, який на першому етапі був дуже схожий на технологію командного рядка з наступними відмінностями:

при відображенні символів з метою підвищення виразності зображення було дозволено виділити частину символів кольором, зворотним зображенням, підкресленням і мерехтіння; курсор може бути представлений певною вибраною областю

колір і охоплює кілька символів і навіть частину екрану; реакція на

натискання будь-якої клавіші багато в чому почала залежати від

з яких частин курсора знаходиться в.

Крім часто використовуваних клавіш, стали використовуватися такі маніпулятори, як миша, трекбол і т.д., що дозволило швидко вибрати потрібну область екрану і перемістити курсор; широке використання кольорових моніторів.

Зовнішній вигляд простого графічного інтерфейсу збігається з широким використанням операційної системи MS DOS. Типовим прикладом його використання є оболонка файлу Norton Commander і текстові редактори MaltiEdit, ChiWriter, Microsoft Word для DOS, Лексикон і т.д.

*Full WIMP-інтерфейс,*став другим етапом розробки графічного інтерфейсу, який характеризується наступними особливостями:

вся робота з програмами, файлами та документами відбувається в вікна; програми, файли, документи, пристрої та інші об'єкти представлені у вигляді значків (іконок), які перетворюються в вікна при відкриваній; всі дії з об'єктами здійснюються за допомогою меню, яке

Стає основним контролем. маніпулятор виступає основним

інструментом управління.

Слід зазначити, що WIMP-інтерфейс вимагає для його реалізації підвищеної вимоги до продуктивності комп'ютера, обсягу його пам'яті якісного програмного забезпечення растрового кольорового дисплея, орієнтованого на даний тип інтерфейсу. В даний час wiMP-інтерфейс став де-факто стандартом, а операційна система Microsoft Windows стала яскравим його представником.

Крім того, ***Xerox PARC*** розробила два ***тривимірних інтерфейси:***

конічні дерева; Стіна в перспективі.

Інтерфейс *«конічних дерев»* являє собою візуалізацію файлової системи комп'ютера і схожий на систему дитячих пірамід, кожен рівень яких відповідає рівню файлового каталогу. Самі файли з каталогу відображаються у вигляді 3-мірної каруселі під їх каталогом. Сіль моделі полягає в тому, що потрібний файл можна «наблизити», повернувши каруселі (можливо, не один), що йде в режимі анімації.

Інтерфейс wall-in-perspective також відображає файлову систему, але за межами її ієрархії, за двома параметрами, такими як частота доступу до файлу і його розмір. Це звичайна стіна, тільки дуже довга, розділена на три сегменти. Середина з них відображається плоскою на екрані, а два крайніх йдуть в перспективу.

Природний інтерфейс

Природний інтерфейс заснований на використанні природних (природних) якостей, властивих користувачам. Існують наступні типи природних інтерфейсів: SILK (мова); біометричні (мімік); семантична (публічна).

SILK (Speech, Image, Language, Knowledge) - **це** інтерфейс, заснований на чотирьох поняттях: "мова", "образ", "мова", "знання". Цей інтерфейс найближчий до звичайної людської форми спілкування. Він також перетворює в форму, зрозумілу людині. Даний тип інтерфейсу вимагає великих витрат на апаратне забезпечення, тому знаходиться в стадії розробки і вдосконалення і використовується до цих пір тільки у військових цілях.

Перша *мовна технологія* з'явилася в середині 90-х років після появи недорогих звукових плат і широкого використання технологій розпізнавання мовлення. Команди давалися голосом шляхом вимовляння спеціальних стандартних слів (команд), які повинні вимовлятися чітко, в однаковому темпі з обов'язковими паузами між словами. З огляду на те, що

алгоритми розпізнавання мовлення були недостатньо розроблені, потрібно індивідуальне попереднє налаштування комп'ютерної системи на конкретну. Перейти до користувача.

В даний час ця мовна технологія була розроблена і впроваджена в багатьох програмних продуктах (мовні команди в MAC OS, пошук мовлення в Google, голосовий блокнот і т.д.).

Біометричні технології ("Мімік Інтерфейс") виникли в кінці 90-х років і в даний час знаходяться в стадії розробки. Міміка, напрямок зору, розмір зіниці та інші ознаки людини використовуються для управління комп'ютером. Для ідентифікації користувача використовується малюнок райдужної оболонки ока, відбитки пальців та інша унікальна інформація, яка зчитується з цифрової камери, а потім за допомогою програми розпізнавання зображень з цього зображення я виділяю Команди.

Семантичний (публічний) інтерфейс виник в кінці 70-х років XX століття, з розвитком штучного інтелекту. Його навряд чи можна назвати самостійним типом інтерфейсу, так як він включає і інтерфейс командного рядка, і графічний, і мовний, і імітуючий інтерфейс. Його головною особливістю є відсутність команд при спілкуванні з комп'ютером. Запит формується природною мовою, у вигляді відповідного тексту і зображень. В даний час використовується у військових цілях. Такий інтерфейс вкрай необхідний в атмосфері повітряного бою.

Ефективні властивості інтерфейсу

Для того щоб ІП був якісним (ефективним), він повинен володіти наступними властивостями:

1. *Природність*інтерфейсу, тобто його здатність виробляти повідомлення і результати, які не вимагають додаткових пояснень.
2. *Послідовність*інтерфейсу, тобто його здатність надавати користувачам можливість передавати існуючі знання новим завданням, швидше вивчати нові аспекти, а тому зосередитися на вирішенні завдання, а не витрачати час на з'ясування відмінностей у використанні тих чи інших елементів управління, команд і т.д.

   Послідовність розглядається в трьох аспектах:

узгодженість всередині програми (та ж команда повинна виконувати ті ж функції, де б вона не відбувалася, і таким же чином); послідовність у виробничому середовищі (додаток повинен «спиратися» на знання і навички користувачів, які він отримав раніше при роботі в середовищі ОС); узгодженість у використанні імен (поведінка кожного об'єкта інтерфейсу повинна відповідати присвоєному йому імені).

3. Зручність інтерфейсу, тобто його здатність запобігати ситуаціям, які, ймовірно, закінчаться помилками через неправильне введення команди або даних користувачем.
4. *Оборотність*інтерфейсу, тобто його здатність супроводжувати кожну дію користувача візуальним або аудіо підтвердженням того, що додаток отримав команду.
5. *Простота*інтерфейсу, тобто простота в його використанні, вивченні і в наданні доступу до всього переліку функціональних можливостей, передбачених цією програмою.
6. *Гнучкість (адаптивність)*інтерфейсу, тобто його здатність враховувати рівень підготовки і продуктивність користувача.
7. *Естетична*привабливість, тобто його здатність додатку забезпечувати формування на екрані такого середовища, що не тільки сприяло б розумінню користувачем представленої інформації, але і дозволило б зосередитися на його найважливіших аспектах.

Стандарти технології інтерфейсу

Технологічні стандарти встановлюють зовнішній вигляд графічних елементів інтерфейсу користувача, а також визначають словниковий запас елементів графічного управління. Кожен такий елемент має стандартизовані властивості описуваного виду, що становить його нормативні акти. Порушення цього регламенту слід розцінити як ***помилку в розробці інтерфейсу користувача.***

Прикладами стандартів, що належать до групи технологічних стандартів, є ISO 9241, ISO/IEC 10741, 11581.

Ергономічні стандарти інтерфейсу

Ергономічні стандарти встановлюють характеристики безпеки та продуктивності інтерфейсу користувача.

Основна увага в ергономічних стандартах приділяється якості функціональних характеристик інтерфейсу користувача, до яких відносяться:

виконання завдання; самоопис;

керованість;

дотримання очікувань користувача;

толерантність (толерантність, імунітет) до помилок;

налаштовуваність; навчальність;

практичність (зрозумілість, видимість, навчання, зручність, простота).

Якщо врахувати, що користувальницький інтерфейс має властивості звичайного текстового об'єкта, який взаємодіє з користувачем, то важливо враховувати такі характеристики, як:

Розмір шрифту; колірний дизайн;

навігація через елементи інтерфейсу.

Для комп'ютерного інтерфейсу також важливо враховувати особливості, пов'язані з комфортом презентації екрану, достатньою ефективністю реакції програми на дії користувача, зручністю маніпулювання мишею і клавіатурою (і їх показниками швидкості).

До групи ергономічних стандартів належать ISO 9241, ISO/IEC 13407, 12119, 9126

Принципи розробки користувацьких інтерфейсів

1. Інтерфейс призначений для інтерактивності

Інтерфейси необхідні для взаємодії людини зі світом. Вони допомагають зробити комплекс простим, показати співвідношення елементів, об'єднати нас або відключити нас, задовольнити наші очікування і забезпечити доступ до послуг. Дизайн інтерфейсу має мало спільного з мистецтвом, тому що ефективність інтерфейсу може бути виміряна. З іншого боку, кращі інтерфейси - це більше, ніж вирішена проблема. Вони можуть надихати, захоплювати і занурювати вас в інший світ.

2. Ясність – завдання No 1

Ясність лежить в основі будь-якого інтерфейсу. Люди повинні швидко зрозуміти, з чим вони працюють, навіщо їм це потрібно, і що з цим можна зробити, припустити, від чого будуть їх дії, а потім почати використовувати його. Clarity підживлює впевненість користувачів у своїх діях і робить використання інтерфейсу приємним. Сто кришталево чистих сторінок краще, ніж одна сторінка, яка знаходиться в хаосі через надлишок інформації.

3. Приверніть увагу будь-якою ціною

Ми живемо в світі, де нас часто відволікають і переривають. Важко читати тихо, щоб нас ніщо не відволікало. Увага – золото. Тому завжди пам'ятайте про призначення конкретного екрану. Якщо мета полягає в тому, щоб прочитати статтю, нехай користувач закінчить читати її перед запуском оголошення. Не заповнюйте сторону сторінки всілякими відволікаючими нісенітницями. Пам'ятаючи про увагу, ви не тільки зробите читачів щасливішими, але і підвищите ефективність взаємодії.

4. Дайте користувачеві контроль над ситуацією

Люди відчувають себе впевнено, коли вони контролюють ситуацію. Погано розроблені програми змушують людей робити незаплановані дії, заважати шляху до мети і навіть несподівано закриватися. Інформуйте користувача, періодично показуючи стан системи, пояснюючи, що станеться, якщо ви зробите ту чи інші дії, і даючи уявлення про те, що їх чекає на кожному кроці. Не бійтеся бути занадто очевидними. Очевидність не відбувається багато.

5. Пряма взаємодія краще

Кращий інтерфейс - це його відсутність, тобто можливість безпосередньо взаємодіяти з фізичними об'єктами. Враховуючи, що це не завжди можливо і що об'єкти стають все більш інформаційними, ми створюємо інтерфейси, які допомагають взаємодіяти з ними. Найпростіший спосіб - додати безліч шарів глянцевих кнопок, хрому, графіки, опцій, властивостей, вікон, додатків. Однак користувач буде змушений обробляти їх, а не фактичні об'єкти, на яких виконуються дії. Замість цього створіть інтерфейси, з якими можна впоратися за допомогою жестів, як і в житті. Ідеальний інтерфейс залишає відчуття прямого контакту з об'єктами на екрані.

6. Одна основна дія на екрані

Кожен екран повинен бути розроблений для єдиної дії, яка дійсно важлива на даний момент. Так легше вчитися, простіше у використанні, простіше будувати в міру необхідності. Екрани з двома або більше основними діями швидко стають заплутаними. Так само, як письмова стаття повинна мати одну об'єднуючу ідею, екран повинен мати одну основну дію, для якої вона була створена.

7. Залиште вторинні дії вторинними

На екранах з однією основною дією може бути багато вторинних дій, але вони повинні залишатися такими. Ви пишете не статтю, щоб змусити людей поділитися в Twitter, а читати і розуміти. Зробіть вторинний візуально простіше або навіть покажіть його після завершення основної дії.

8. Забезпечити природний наступний крок

Рідкісні дії виконуються за один крок. Так що подумайте про кроки для кожної дії, яка знаходиться у вашому інтерфейсі. Спрогнозуйте наступну дію і надайте їй інтерфейс. Так само, як і в людських стосунках, дайте нам знати, яким буде наступний крок. Не залишайте людину в очікуванні тільки тому, що вона зробила те, що ви хотіли. Намалюйте йому шлях до мети і ненав'язливо допомагайте на кожному кроці.

9. Форма слідує за функцією

Люди відчувають себе впевнено, коли вони справляються з передбачуваними речами. Коли інші люди, тварини, предмети, програми поводяться так, як ми очікуємо, нам це подобається. Тому важливо спроектувати елементи інтерфейсу так, щоб їх можна було використовувати для прогнозування того, які дії вони виконують. Якщо щось схоже на кнопку, вона повинна працювати як кнопка. Не хизуйтеся основами взаємодії, залиште свій творчий потенціал для більш важливих речей.

10. Значення послідовності

Елементи, які виконують подібні функції, повинні виглядати схожими. Це здається логічним і очевидним. Але в цьому принципі часто забувається протилежний ефект: несхвальні дії повинні вказувати дисимутативні елементи. У спробі створити цілісний інтерфейс дизайнери-початківці ігнорують важливі відмінності, використовуючи обмежений набір візуальних елементів.

11. Найкраще працюють суворі візуальні ієрархії

Сувору візуальну ієрархію можна простежити там, де на екрані є чітка послідовність елементів. Тобто порядок розташування подібних елементів однаковий на всіх однакових екранах. Якщо ієрархії немає, то користувачеві складно зрозуміти, де шукати, що потрібно. Мало хто помічає цю ієрархію, але це один з найпростіших способів зробити інтерфейс більш зрозумілим.

12. Порядок розвантажує мозок

Як сказав Джон Маеда у своїй книжці « Простота », порядок в розташуванні елементів може дати відчуття, що багато речей виглядають як малі. Це допомагає людям краще і швидше зрозуміти ваш інтерфейс, оскільки ви показуєте невидимі зв'язки між елементами. Групуйте схожі елементи, підкреслюй зв'язки з розташуванням і візуальною ієрархією. Роблячи це, ви допомагаєте користувачеві, який не повинен розуміти ваш інтерфейс, просто тому, що ви це зробили. Не змушують користувача думати, і він буде вам вдячний.

13. Використовуйте колір, щоб виділити, а не мати сенсу

Колір фізичних об'єктів варіюється в залежності від освітлення. У світлі дня ми бачимо дерево з усіма його деталями і відтінками, але якщо ми подивимося на нього проти заходу сонця, то побачимо тільки чорний контур. Так само, як і у фізичному світі, де колір об'єкта може сильно відрізнятися, колір не повинен бути основним в інтерфейсі. Він може допомогти, коли потрібно спрямувати увагу на певний елемент, але не використовуйте його як єдиний спосіб розрізнити елементи. Використовуйте світлі або приглушені фонові кольори, залишаючи яскраві для акцентуації. Звичайно, можна використовувати яскраві кольори, але тільки тоді, коли ви впевнені, що це буде позитивно сприйнято вашою аудиторією.

14. Поступовий зовнішній вигляд

Показувати лише те, що вам потрібно на поточному екрані. Якщо людям доводиться робити вибір, дайте їм достатньо інформації для цього, і пориньте в деталі на наступному екрані. Уникайте звички говорити і показувати все відразу. Коли це можливо, розділіть кілька рішень на різні екрани. Це допоможе зберегти ясність у взаємодії.

15. Допоможіть користувачам на цьому шляху

В ідеальних інтерфейсах допомога не потрібна, тому що вони зрозумілі і прості у використанні. У реальному житті інтерфейси іноді потребують допомоги, але вона повинна бути надана частинами і там, де вона дійсно потрібна, приховуючи її до кінця часу. Ви можете попросити людей зайти в розділ довідки і знайти там відповідь на їх питання, але краще вбудувати поради там, де вони потрібні. Просто переконайтеся, що користувачі знайомі з такими інтерфейсами.

16. Вирішальний момент: нульовий крок

Перший досвід взаємодії з інтерфейсом має вирішальне значення, що часто недооцінюють дизайнери. Труднощі з інтерфейсом найчастіше проявляються на цьому етапі. Щоб допомогти користувачам швидко зрозуміти інтерфейс, створіть довідкову сторінку перед завантаженням основного вмісту. Розкажіть нам і покажіть, що і як вам це потрібно зробити. Коли користувачі розуміють правила, їм сподобається користуватися інтерфейсом.

17. Існуючі проблеми є найважливішими

Люди шукають рішення своїх поточних проблем, а не потенційні проблеми, які чекають їх у майбутньому. Тому не варто проектувати інтерфейс, заточений для гіпотетичних завдань. Досліджуйте поточні проблеми та створюйте для них інтерфейс. Це не так захоплююче, але це, безумовно, принесе плоди, коли реальні користувачі почнуть використовувати ваш інтерфейс.

18. Кращий дизайн невидимий

Парадоксально, але факт: великий дизайн невидимий. Причина в тому, що користувач може зосередитися на вирішенні своїх проблем, а не мати справу з інтерфейсом. Коли користувач успішно виконав своє завдання, він щасливий і не бачить причин дякувати інтерфейсу за це. Можливо, це звучить невтішно для дизайнерів, адже ніхто не скаже, наскільки хороший ваш дизайн. Однак великі дизайнери знають, що щасливі користувачі мовчать.

19. Черпають натхнення з інших областей

Дизайн, типографія, авторське право, інформаційна архітектура та візуалізація. Всі ці області є частиною інтерфейсу. Вони можуть бути вторинними або основними у вашому інтерфейсі. Не зосередьтеся на

дизайні, розширюйте свої горизонти і шукайте натхнення в принципах інших дисциплін. Подумайте про те, що ви можете дізнатися з написання коду, правил дизайну книг, скейтбордингу або карате.

20. Інтерфейси створюються для використання

Як і в інших типах дизайну, дизайн є успішним, коли люди користуються ним. Як красивий стілець, на якому незручно сидіти, інтерфейс вважається збоєм, коли користувачі не хочуть ним користуватися. Тому важливо пам'ятати, що дизайн інтерфейсу поєднує в собі як створення середовища для вирішення проблем, так і продукт, який ви хочете використовувати. Інтерфейс, який задовольняє тільки его дизайнера, не може вважатися хорошим. Інтерфейс хороший, коли він активно використовується.

Порядок виконання лабораторних робіт

1. Визначте вимоги до інтерфейсу користувача IP.
2. Створення профілів потенційних користувачів програмного забезпечення інформаційної системи в табличному вигляді.
3. Визначити функціонал додатку на основі цілей і завдань користувачів
4. Створення кількох сценаріїв користувача для спеціальних профілів користувачів
5. Виділіть вимоги до інтерфейсу користувача на основі сценаріїв і групування операцій:

   основний і підменю (текстовий і графічний опис), кількість сторінок (для веб-систем) і їх призначення, структура основної і підсторінки, кількість і структура діалогових вікон,

   кількість і структура системних повідомлень (помилки, завершення процесу і т.д.) наявність зворотного зв'язку і т.д.
6. Пріоритетні вимоги (необхідні, бажані, додаткові)
7. Виділіть основні вимоги до дизайну.
8. Створіть схему інтерфейсу користувача.
9. Розробка макетів дизайну головної сторінки системи, додаткових сторінок (вкладок), діалогових вікон і повідомлень
10. Обґрунтуйте вибір дизайну.

Питання безпеки

1. Концепція інтерфейсу користувача.
1. Типи користувацьких інтерфейсів.
2. Властивості інтерфейсу користувача.
3. Стандарти інтерфейсу користувача.
4. Принципи розробки інтерфейсу користувача.
5. Етапи розробки інтерфейсу користувача.

ЛАБОРАТОРНА РОБОТА №5
"РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ"

Цілі роботи:

1. Закріплення існуючих знань про засоби розробки програмного забезпечення інформаційних систем.
2. Набуття навичок у сучасних інтегрованих середовищах розробки програмного забезпечення.
3. Набуття навичок у розробці програмного забезпечення для ІВ клієнта.

Завдання роботи:

1. Розробка прототипу програми.
2. Розробіть код програми.
3. Налагодження розробленого додатка.
4. Розробити документ Посібника користувача, який описує мету та функціональність програми

Теоретична інформація

CASE-технологія - це сукупність методологій аналізу, проектування, розробки та обслуговування складних програмних систем, що підтримується комплексом взаємопов'язаних програмних засобів автоматизації. Основою CASEtechnology є використання єдиної бази даних (репозиторію) для зберігання всієї інформації, яка може бути використана в процесі створення системи. У репозиторії можуть зберігатися об'єкти різних типів: структурні діаграми, ескізи екранних форм, моделі даних, опис алгоритмів обробки даних і т. д. Інструменти CASE - це програмні інструменти, які підтримують процеси створення та підтримки ІП, включаючи аналіз і формулювання вимог, проектування прикладного програмного забезпечення і баз даних, генерацію коду, тестування, документацію, забезпечення якості, управління конфігурацією, управління проектами і т.д.

CASE-інструменти включають в себе будь-який програмний продукт, який має наступні основні характерні особливості:

1. наявність потужних графічних інструментів для опису та документування IP;
2. інтеграція окремих компонентів CASE-інструментів, забезпечення керованості процесу розробки IB;
3. використання спеціально організованого сховища метаданих проекту (репозиторію).

Швидка розробка додатків (RAD) є однією з сучасних методологій розробки програмного забезпечення. А також інші методології (MSF, RUP і т.д.) RAD описує ітераційний підхід до організації процесу розробки програмного забезпечення та відповідну модель життєвого циклу. Методологія Rad також часто пов'язана з технологією візуального програмування та використанням сучасних інтегрованих середовищ розробки програмного забезпечення. Методологія RAD заснована на візуалізації процесу створення коду додатку і підтримується інструментальним програмним забезпеченням, яке надає розробникам інструменти візуального програмування. Використання інструментів візуального програмування дозволяє значно прискорити процес розробки додатків, а також знизити трудомісткість роботи над модифікацією вже готової програми, внесенням до неї необхідних доповнень або змін. Інструменти швидкого розробки додатків, як правило, засновані на об'єктно-орієнтованій архітектурі компонентів.

Процедура розробки інтерфейсу за допомогою RAD зводиться до набору послідовних операцій, в тому числі:

1) розміщення компонентів інтерфейсу в потрібному
місці; 2) встановлення моментів часу їх появи на екрані;
3) налаштування пов'язаних атрибутів і подій.

Інтегроване середовище розробки (ISM) - це засіб, за допомогою якого програми розробляються, програмуються, тестуються та налагоджуються. Прикладами сучасних ISR, які підтримують методологію RAD і технологію візуального програмування, є Microsoft Visual Studio, Embarcadero RAD Studio, IntelliJ IDEA, MonoDevelop і т.д.

Порядок виконання лабораторних робіт

1.Розробка програмних алгоритмів.

2. Створіть дерево діалогу.

3.На основі уточнених і уточнених алгоритмів розробити структурну схему застосування.

1. Розробити функціональну схему застосування.

Питання безпеки

1.Які етапи розробки програмного забезпечення.

2.Що таке програмне забезпечення?

3.Перерахує компоненти технічного проекту.

4.Охарактеризуйте структурний підхід до програмування.

1. ЛАБОРАТОРНА РОБОТА No6 "ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ"

6. Тестування програмного забезпечення

Мета роботи: вивчення основних методів тестування програмних засобів, отримання навичок тестування програмного забезпечення.

Завдання роботи:

1. Вивчити основні методи тестування програмних систем.
2. Сформувати єдину ідею реалізованого програмного забезпечення.
3. Визначити розбіжності між функціональністю реалізованого програмного забезпечення і ТК.

Теоретична інформація

Тестування програмного забезпечення - це діяльність, що виконується з оцінки та підвищення якості програмного забезпечення. Ця діяльність, в цілому, заснована на виявленні дефектів і проблем в програмних системах. Тестування програмних систем полягає в динамічній перевірці поведінки програм на остаточному (обмеженому) ***наборі тестів*** (наборі тестових випадків), відібраних відповідно з зазвичай виконуваних дій області застосування і забезпечення верифікації відповідність очікуваній поведінці системи. [СИВОК].

Тестування - запуск виконуваного коду з тестовими даними і вивчення вихідних даних і експлуатаційних характеристик програмного продукту для перевірки правильної роботи системи.

Тестування - це процес запуску своїх програм на певному наборі даних, для якого результат програми відомий заздалегідь або відомі правила поведінки цих програм.

Тестування - це перевірка роботи програм з даними, подібними до реальних, які будуть оброблятися під час роботи системи.

Об'єктиви тестування: Тестування зазвичай проводиться під час розробки та обслуговування на різних рівнях.

Мета тесту. Тестування проводиться відповідно до конкретних цілей (можуть бути визначені явно або неявно) і різних рівнів точності. Визначення мети точним, кількісним способом дозволяє контролювати результати тестування. існують тести, де кількісні параметри і результати випробувань можуть лише побічно свідчити про задоволення цілей тестування (наприклад, «юзабіліті» - легкість, простота використання, в більшості випадків не можуть бути чітко описані кількісними характеристиками).

До *об'єктів* тестування відносяться:

1. Модульне тестування Цей рівень тестування дозволяє перевірити функціонування одного елемента системи. Що вважається елементом - модуль системи визначається контекстом.
2. Інтеграційне тестування. Цей рівень тестування є процесом перевірки взаємодії програмних компонентів/модулів. Інтеграційне тестування - це постійна діяльність, яка передбачає роботу на досить високому рівні абстракції. Найбільш успішна практика інтеграційного тестування базується на інкрементному підході.
3. Тестування системи Охоплює всю систему. Більшість функціональних збоїв повинні бути визначені на рівні модульних та інтеграційних тестів.

Цілями тестування є:

Акцепт/кваліфікаційне тестування Перевіряє поведінку системи на виконання вимог замовника. Це можливо, якщо замовник бере на себе відповідальність за проведення такої роботи.

Тестування інсталяції Щоб перевірити процедуру інсталяції в цільовому середовищі.

Альфа-і бета-тестування: Як мінімум, альфа (внутрішня пробна версія) і бета-версія (пробне використання з вибраними зовнішніми користувачами) версії повинні бути випущені до виходу програмного забезпечення.

Тестування відповідності / Функціональне тестування / Тестування правильності: Переконайтеся, що система відповідає вимогам, описаним на рівні поведінкової специфікації.

Досягнення надійності та оцінювання)

Регресійне тестування тесту програмного забезпечення, призначеного для виявлення помилок у вже протестованих частинах вихідного коду. Такі помилки, які виникають при внесенні змін до програми, які повинні були працювати, називаються *помилками регресії.*
(англ. *регресійні помилки*).

Визначення успішності регресійних тестів (IEEE 610-90 «Стандартний глосарій програмної інженерії термінології»): «повторне випадкове тестування системи або компонента для перевірки внесених змін не повинно призводити до ненавмисних ефектів».

Тестування продуктивності Спроба досягає кількісних меж через характеристики самої системи та її робочого середовища.

Стрес-тестування З метою досягнення своїх реальних /досяжних (т продуктивності) експлуатаційних можливостей і збільшення навантаження, аж до досягнення запланованих характеристик і далі, з моніторингом поведінки протягом усього збільшення навантаження системи.

Сравнительное тестирование (Back-to-back testing)

Восстановительные тесты (Recovery testing)

Конфигурационное тестирование (Configuration testing)

Юзабіліті тестування Мета полягає в тому, щоб перевірити, наскільки легко кінцевий користувач системи може освоїти її, включаючи не тільки функціональний компонент – саму систему, але і її документацію; наскільки ефективно користувач може виконувати завдання, автоматизовані за допомогою цієї системи; нарешті, наскільки добре система застрахована (з точки зору потенційних збоїв) від помилок користувачів.

Разработка, управляемая тестированием (Test-driven development)

Як правило, цілі тестування можна визначити наступним чином:

1. **Тестування на дефект** проводиться для виявлення невідповідностей між програмним забезпеченням та його специфікацією, які викликані

помилками або дефектами. Такі тести призначені для виявлення помилок в системі, а не для імітації його роботи. Метою тестування дефектів є виявлення прихованих дефектів у програмній системі до його передачі замовнику. і повинен правильно працювати з усіма вказаними тестовими даними. При тестуванні дефектів проводиться тест, який викликає неправильну роботу програми і, отже, виявляє дефект. Зверніть увагу на цю важливу особливість: тестування дефектів демонструє наявність, а не відсутність дефектів у програмі.

2. **Статистичне тестування** оцінює продуктивність і надійність програмного забезпечення, а також роботу в різних режимах роботи. Тести призначені для моделювання фактичної роботи системи з реальними вхідними даними.

Структурне тестування

Метод передбачає створення тестів на основі структури **системи** і її реалізації.

Такий підхід іноді називають тестуванням *білих коробок,* "скляною коробкою" або "прозорою коробкою", яка дозволяє зазирнути всередину програмного забезпечення.

Як правило, структурне тестування застосовується до відносно невеликих програмних елементів, таких як підпрограми або методи, пов'язані з об'єктами. Наприклад, ви можете визначити з аналізу коду, скільки контрольних тестів потрібно запустити, щоб усі оператори запускав принаймні один раз під час процесу тестування.

Функціональне тестування

Метод тестування заснований на тому, що всі тести засновані на перевірці не самого програмного забезпечення, а тільки виконуваних ним функцій.

Програма представлена у *зв'складі «чорного ящика» (Black-box),*поведінку якого можна визначити тільки вивчаючи його вхідні і відповідні вихіднідані.

Комбіноване тестування

Метод поєднує в собі перевірку як структури програмного забезпечення, так і його функціонування.

Такий підхід називається тестуванням*сірої коробки* (Тестування*сірої коробки).*

Тестові заходи:

1. **Планирование** (Planning)
2. **Генерация сценариев тестирования** (Test-case generation).

Створення тестових сценаріїв базується на рівні та конкретних методах тестування.

3. **Разработка тестового окружения** (Test environment development).

Навколишнє середовище повинно забезпечити розробку та контроль тестових сценаріїв, тестового журналювання та можливість відновлення очікуваних та відстежуваних результатів випробувань, самих сценаріїв та інших тестових активів.

4. Виконання: Виконання випробувань повинно містити основні принципи проведення наукового експерименту:

вся робота і результати процесу повинні бути записані

тестування; форма журналювання таких творів та їх результати повинні бути такими, щоб відповідний зміст був зрозумілим, однозначно інтерпретованим і повтореним іншими особами; тестування має проводитися відповідно до зазначених та

документовані процедури; тестування повинно бути зроблено над однозначно ідентифікованим.

версія та конфігурація програмної системи

5. **Анализ** результатов тестирования (Test results evaluation).
6. Звітування про проблеми/тест

журнал)

7. Дефекти відстеження дефектів можуть (і найчастіше повинні) бути проаналізовані, щоб визначити, коли і де дефект вперше з'являється в системі, які типи помилок викликали ці дефекти.

Тестові документи

1. **План випробувань - це** організаційний документ, який містить вимоги до того, як тестування повинно виконуватися в даному проекті.

   Створений менеджером тестів.

2. **Вимоги до тестування** - це документи, які деталізують, які аспекти поведінки системи повинні бути перевірені.

Створена розробниками процедури тестування. 3. **Плани випробувань** (тестові**специфікації) - документи,**які містятьдокладний покроковий опис того, як повинні бути перевірені вимоги до тестування.

Створений розробниками тестів.

1. **Звіти про перебіг тестування** (які можуть бути створені автоматично або вручну), які містять інформацію про невідповідність, виявлені в результаті тестування.

   Створений тестувальниками.

2. **Проблемні** звіти - це документи, які відправляються в команду розробників для аналізу з метою визначення причини невідповідності.

План випробувань

Структура тестового плану може відповідати структурі тестових вимог або слідувати логіці зовнішньої поведінки системи. Кожен пункт тестового плану описує, як перевіряється правильне функціонування реалізації програмного забезпечення, і містить:

посилання на вимогу(і), яка перевіряється цим пунктом; конкретний **вхідний ефект** на програму (вхідні значення)
даних); **очікувана реакція** програми (тексти повідомлень, значення результатів);

опис **послідовності дій, необхідних**для виконанняпунктів тестового плану.

Можливі форми підготовки тестових планів:

1. У вигляді текстових документів, в яких окремі розділи є описами тестових кейсів, кожен приклад включає в себе список послідовності

дій, які необхідно виконати - **тестові сценарії,**а також очікувані відповіді системи на ці дії.

2. Для автоматизованого тестування скрипт може бути **написаний офіційною мовою.**
3. **Таблиця** використовується для чітко визначених і формально визначених потоків вхідних даних в системі. Наприклад, кожен стовпець таблиці може бути тестовим прикладом, кожен рядок може бути описом вхідного потоку даних, а комірка таблиці записує значення, передане в цьому тестовому прикладі цьому потоку.
4. У вигляді **державної машини** він використовується при тестуванні протоколів зв'язку або програмних модулів, взаємодія яких із зовнішнім світом здійснюється за допомогою обміну повідомленнями на заданому інтерфейсі.

Порядок виконання лабораторних робіт

1. Сформувати сценарій системного тестування розробленого програмного забезпечення.
2. Вкажіть специфікації для тестування окремих компонентів (функцій).
3. Опишіть стан середовища (введення) і очікувану послідовність подій в системі (очікуваний результат в специфікації для кожного тестового випадку.
4. Розробка тестів.
5. Перевірте загальний стан здоров'я та індивідуальну функціональність розробленого додатку.
6. Виправте можливі помилки в додатку.
7. Виконати перевірку функціональності розробленого додатку, порівнявши їх з існуючим переліком функціональних вимог.

Контрольні питання

1. Концепція тестування програмного забезпечення.
2. Завдання тестування.
3. Види тестування.

4. Що таке тестовий сценарій?
5. Які види тестування необхідно використовувати в процесі розробки складного програмного забезпечення?
6. Модульне тестування. Концепція модуля.
7. V-подібна модель. Статичне і динамічне тестування.
8. Валідація та верифікація. Тестування методом «чорного» і «білого» ящика.
9. Тестовий випадок і тестове покриття.