Державний вищий навчальний заклад
«Запорізький національний університет»
Міністерства освіти і науки України

Тодоріко О.О.

ТЕХНОЛОГІЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Методичні вказівки
до виконання лабораторних робіт
для студентів II курсу математичного
факультету

Затверджено
вченою радою ЗНУ
Протокол №__ від.____.2009р.

Запоріжжя 2009

УДК 519.85:004.42(076)

Тодоріко О.О. Технологія розробки програмного забезпечення: Методичні вказівки до виконання лабораторних робіт для студентів II курсу математичного факультету:– Запоріжжя: ЗНУ, 2009. – 40с

Методичні вказівки містять 4 лабораторні роботи з чотирьох тем: «Організація розробки програмного забезпечення», «Документація процесу розробки програмного забезпечення», «Автоматизація розробки програмного забезпечення», «Проектування».

Методичні вказівки корисні студентам спеціальності «Інформатика», а також студентам різних спеціальностей для одержання основних навичок по технології розробки програмного забезпечення.

Рецензент В.А. Єрмолаєв
Відповідальний за випуск С.Ю. Борю

Вступ

В курсі «Технологія розробки програмного забеспечення» розглядаються наступні теми:

1. Організація розробки програмного забезпечення
2. Документація процесу розробки програмного забезпечення
3. Автоматизація розробки програмного забезпечення
4. Проектування

Ці методичні вказівки дозволять отримати такі знання та вміння:

1. Аналізувати предметну область та вимоги до ПЗ;
2. Будувати моделі проблемних областей у відповідності до задачі що вирішується;
3. Будувати архітектуру ПЗ у відповідності до задачі що вирішується;
4. Будувати інтерфейс користувача у відповідності до задачі що вирішується;
5. Розробляти ПЗ в команді розробників;
6. Розробляти документацію процесу розробки ПЗ.

Методичні вказівки містять 4 лабораторні роботи з використанням відповідного до завдання програмного забезпечення, яке обирається студентом самостійно чи за допомогою викладача.

Задачі курсу. Оволодіти навичками розробки великих програмних комплексів. Засвоїти всі етапи розробки програмного забезпечення.

У результаті вивчення курсу студент повинен

Знати:

– моделі життєвого циклу програмного забезпечення;
– методи аналізу предметної області та вимог до ПЗ;
– методи контролю якості ПЗ;
– методи аналізу та побудови архітектури ПЗ;
– принципи побудови інтерфейсу користувача;
– задачі управління розробкой ПЗ.

Вміти:

– будувати моделі проблемних областей у відповідності до задачи що вирішується;
– будувати архітектуру ПЗ у відповідності до задачі що

вирішується;

– розробляти ПЗ в команді розробників

– розробляти відповідну документацію до розробленого ПЗ.

Структура методичних вказівок

Методичні вказівки містять теоретичний матеріал з чотирьох тем, що дозволяє одержати навички по технології розробки програмного забезпечення.

Друга частина методичних вказівок містить завдання на 4 лабораторні роботи. Для виконання кожної лабораторної роботи необхідно прочитати теоретичний матеріал.

Вимоги до виконання лабораторних робіт

При виконанні кожної роботи необхідно:

1. Уважно прочитати завдання та доповнити його.
2. Прочитати відповідний теоретичний матеріал.
3. Виконати практичну частину лабораторної роботи на комп'ютері згідно до завдання.
4. Впродовж виконання пунктів завдання обов'язково докладно занотувати послідовність дій.
5. Звіт з лабораторної роботи оформляється згідно з стандартними вимогами до оформлення звітів і повинен містити:
   - номер лабораторної роботи;
   - назву лабораторної роботи;
   - порядок виконання лабораторної роботи;
   - всі пункти, передбачені завданням;
   - електронний варіант програмного продукту, якщо він передбачен завданням.
6. Після здачі практичної частини студент зобов'язаний подати викладачеві звіт про виконання лабораторної роботи і захистити її.

1 Організація розробки програмного забезпечення

Серйозні програмні продукти рідко розробляються однією людиною. Зазвичай цим займаються групи людей, іноді досить численні.

1.1 Група проекту та ролі учасників групи

Пов'язані з життєвим циклом процеси різноманітні по своїй суті, не кажучи про те, що самі проекти дуже важкі, включають багато окремих потребуючих програмного рішення завдань і припускають наявність спеціальних знань. Можна називати співтовариства людей, що працюють над проектом, по-різному: група, підрозділ, бригада або команда, але місія одна - створення якісного продукту в умовах обмежень на час і ресурси.

Проектна група *(development team)* – співтовариство працюючих над проектом людей, об'єднаних з метою створення якісного програмного продукту в обмежений термін у рамках відведеного бюджету.

У такій групі в кожного співробітника своя роль. Існує стандартний набір функцій, які виконуються членами проектної групи. Для простоти передбачається, що кожна роль належить окремому співробітнику. На практиці в більшості організацій співробітники часто виконують по кілька функцій.

1.1.1 Керівник проекту - Project manager

Software development manager, Project manager, Producer.

Керівник проекту відповідає за розробку якісного продукту в строк і в рамках установленого бюджету.

Основні види діяльності:

- планування роботи й складання бюджету;
- прийняття критичних для ходу проекту рішень;
- ведення графіка проекту й забезпечення дотримання строків;
- координування робіт (керування взаєминами) проектної групи;
- керування функціональними специфікаціями (технічними завданнями).

1.1.2 Менеджер по маркетингу - Product manager

Product marketing manager.

Менеджер по маркетингу відповідає за відповідність продукту фірмовому стилю своєї компанії й за маркетингову діяльність після випуску продукту.

Питання, пов'язані з рекламою, поширенням нових версій, навчанням продавців і дилерів, розрахунком рентабельності продукту, визначенням потреб ринку програмних продуктів і т.п. (тобто маркетингом), у вивчення даної дисципліни не входять.

1.1.3 Розроблювач - Developer

Розроблювачів, як правило, буває мало.

Розроблювачі відповідають за виконання вимог замовника.

Розроблювач архітектури - Architect

Архітектор відповідає за архітектуру продукту, що зрозуміла, проста, ефективна, надійна й піддається модифікації.

Завдання архітектора:

- Визначення **загальної** внутрішньої **архітектури** коду й даних.
- Визначення **принципів** обміну даними між зв'язаними компонентами.
- Визначення **стратегії** розробки повторно використовуваних компонентів.
- Складання **плану тестування** “чорного ящика” **на найвищому рівні.**
- Розробка тестів, що перевіряють відповідність коду технічному завданню.

Аналітик предметної області - Software analyst.

Аналітик відповідає за правильність розуміння розв'язуваних завдань.

Завдання аналітика:

- Забезпечення зв'язку між замовником і проектною групою.
- Розуміння того, що хочуть користувачі.
- Розуміння того, як виразити бажання користувача в термінах, зрозумілих учасникам проектної групи.
- Розробка логіки роботи програмного продукту.

- Складання робочих специфікацій (детальних технічних завдань).

Фахівець із інтерфейсу користувача - User interface designer

Розроблювач інтерфейсу відповідає за зручність і функціональну відповідність користувальницького інтерфейсу.

Звичайно інтерфейс користувача створює збалансована група з фахівців із психології, ергономіці, комп'ютерній графіці й програмуванню, але всю повноту відповідальності несе тільки одна людина.

Програміст - Programmer

Програміст відповідає за надійний і повний продукт.

Програмісти займаються розробкою певної частини продукту, що належить до внутрішньої архітектури.

Завдання програміста:

- Створення продукту, що задовольняє технічному завданню.
- Тестування продукту (забезпечення якості коду).
- Забезпечення термінів розробки коду.
- Забезпечення установки програмного продукту.

1.1.4 Тестер - Tester

Тестер відповідає за погоджений і надійний продукт.

Тестер може входити до групи розроблювачів. Тестер забезпечує те, що всі особливості продукту будуть відомі до випуску завершальної версії.

Завдання тестера:

- Розробка стратегії й плану тестування кожного етапу життєвого циклу.
- Верифікація на кожному етапі життєвого циклу (деталей продукту, використовуваної термінології, функціональних специфікацій, коду, користувальницького інтерфейсу).
- Комплексне тестування проекту.
- Документування результатів тестування.

1.1.5 Технічний письменник - Writer

Технічний письменник відповідає за продукт, який можна використовувати й супроводжувати.

Завдання представників групи документування:

- Розробка документації (різноманітні інструкції й керівництва), що допомагає зробити програмне забезпечення зрозумілим для користувачів і всіх розроблювачів.
- Ведення предметного покажчика (визначення єдиної термінології).
- Навчання користувачів (відповіді на питання й надання необхідної інформації).

1.1.6 Представник групи технічної підтримки - Logistic

Technical support.

Логістик відповідає за гладке впровадження продукту.

Це може бути одна людина або керівник цілої групи співробітників, що безпосередньо контактує з користувачами.

Завдання представника групи технічної підтримки:

- Забезпечення готовності замовника до впровадження (контроль своєчасності виконання підготовчих робіт, перевірка наявності необхідної інфраструктури).
- Адміністрування локальної мережі (якщо необхідно).
- Відстеження запитів на розширення функціональних можливостей.

1.1.7 Інші фахівці

У розробці проектів можуть брати участь фахівці з баз даних, захисту інформації, мереж, апаратного забезпечення, надійності, ергономіки (вони знають, як спроектувати зручний і корисний продукт), експерти й інженери по знаннях, а також юристи, бухгалтери й т.д.

1.2 Модель проектної групи

Під моделлю проектної групи розуміють структуру, що дозволяє відслідковувати постійно мінливі вимоги в проекті, що веде група проекту.

Модель групи проекту (рис.1.1):

- уводить поняття ролі для всіх учасників групи;
- дає чітке визначення зон відповідальності кожної ролі;

- визначає активність ролі протягом усього життєвого циклу.

Модель побудована навколо ідеї маленької групи учасників, що працюють у тісному взаємозв'язку, кожний з яких виконує свої ролі. У моделі визначені шість ролей:

1. Менеджер по маркетингу – *Product manager*.
2. Керівник проекту – *Program manager*.
3. Розроблювач – *Developer*.
4. Тестер – *Tester*.
5. Технічний письменник – *Writer*.
6. Логістик – *Logistic*.

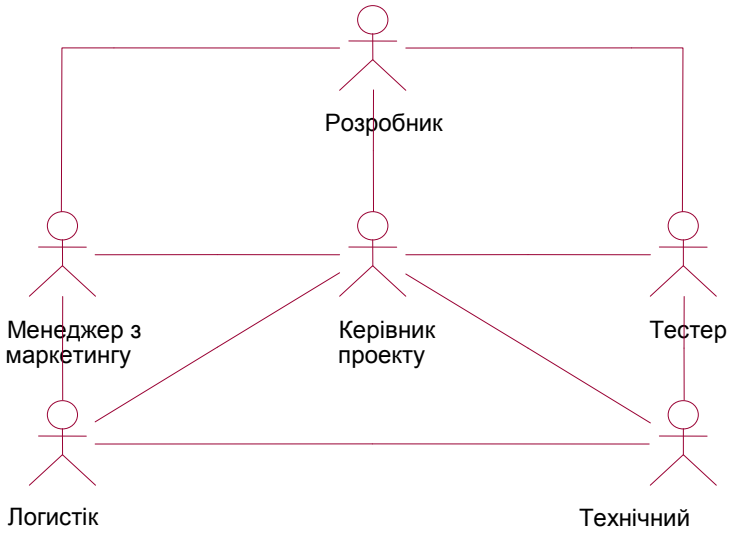


Рис. 1.1 – Модель групи проекту

Характеристики групи проекту:

1. Спілкування кожного з кожним, причому кожний робить реальну роботу.
2. Загальні для всіх членів групи мети й плани.
3. Кожний розуміє як проблеми користувача, так і проблеми розроблювача.
4. Кожний відповідає за свою роботу, у тому числі й перед групою.

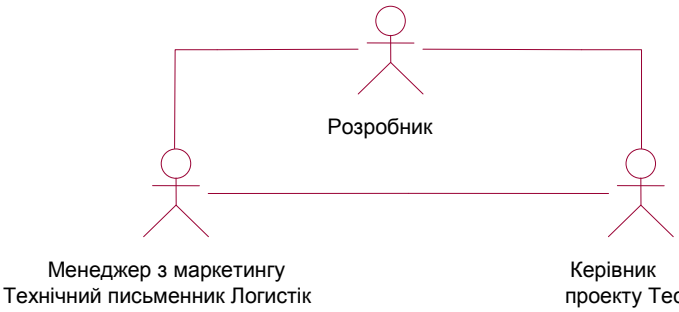
У цій моделі відсутня ієрархія учасників групи, тому що питання “хто кому підпорядкований” втрачає смисл. Однак існує кілька моментів, на які також варто звертати увагу:

- більша чисельність групи проекту може вимагати дуже багато часу на спілкування, щоб реалізувати характеристику під номером 1;
- вище керівництво має обмежений контроль над процесами усередині групи;
- учасники групи повинні повністю розуміти й приймати свої ролі.

Для дуже більших проектів склад розроблювачів розбивається на підгрупи, діяльність яких координується. Підгрупи можуть бути функціональними, або, у свою чергу, теж можуть бути розбиті на ще менші підгрупи. Це дає можливість організувати паралельну роботу над проектом. Кожна група від найвищого до найнижчого рівня складається не більш ніж з п'яти-семи чоловік. Групи працюють у тісному контакті один з одним і забезпечують процеси життєвого циклу.

Слід зазначити, що модель групи проекту ніяк не співвідноситься з організаційною структурою організації-розроблювача. На практиці часті випадки, коли в одній проектній групі працюють люди з різних організацій, що підкоряються різним керівникам. За кожним членом проектної групи закріплюється конкретна роль, для якої будується специфічний план робіт, що потім входить у загальний план проекту як складова частина.

Для невеликих проектів деякі ролі можуть бути сполучені, і



можуть виконуватися однією людиною (рис 1.2):

Рис. 1.2 – Модель групи проекту, де учасники виконують кілька ролей одночасно

Як правило, в основний склад проектної групи входять аналітик, розроблювач і тестер.

Інші ролі не менш важливі, однак виконуючі ці ролі учасники можуть одночасно працювати над декількома проектами одночасно (технічний письменник, логістик, менеджер по маркетингу).

Кількість учасників визначається як складністю проекту, так і масштабами організації (наприклад, в *Microsoft* на кожного розроблювача по двох тестера). Природно, що кожна організація-розроблювач має свою модель групи проекту, і свою методологію створення програмних продуктів, які в остаточному підсумку й визначають корпоративну культуру організації.

2 Документація процесу розробки програмного забезпечення

При розробці програмного засобу (ПЗ) створюється великий обсяг різноманітної документації. Вона необхідна:

- як засіб передачі інформації між розроблювачами ПЗ,
- як засіб керування розробкою ПЗ,
- як засіб передачі користувачам інформації, необхідної для застосування й супроводу ПЗ.

На створення цієї документації приходиться більша частка вартості ПЗ.

Документацію процесу розробки можна розбити на дві групи:

1. Документи керування розробкою ПЗ – *process documentation.*
2. Документи, що входять до складу ПЗ – *product documentation.*

Документи керування розробкою ПЗ

Протоколюють процеси розробки й супроводу ПЗ, забезпечуючи зв'язку усередині колективу розроблювачів і між колективом розроблювачів і менеджерами *(managers)* – особами, що управляють розробкою. Ці документи можуть бути наступних типів:

- Плани, оцінки, розклади. Ці документи створюються менеджерами для прогнозування й керування процесами розробки й супроводу.
- Звіти використання ресурсів у процесі розробки. Створюються менеджерами.

- Стандарти. Ці документи пропонують розроблювачам, яким принципам, правилам, угодам вони повинні випливати в процесі розробки ПЗ. Ці стандарти можуть бути як міжнародними або національними, так і спеціально створеними для організації, у якій ведеться розробка даного ПЗ.
- Робочі документи. Це основні технічні документи, що забезпечують зв'язок між розроблювачами. Вони містять фіксацію ідей і проблем, що виникають у процесі розробки, опис використовуваних стратегій і підходів, а також робітники (тимчасові) версії документів, які повинні ввійти в ПЗ.
- Замітки й переписка. Ці документи фіксують різні деталі взаємодії між менеджерами й розроблювачами.

Документи, що входять до складу ПЗ

Описують програми ПЗ як з погляду їхнього застосування користувачами, так і з погляду їхніх розроблювачів і супровідників (відповідно до призначення ПЗ). Тут слід зазначити, що ці документи будуть використатися не тільки на стадії експлуатації ПЗ (у її фазах застосування й супроводу), але й на стадії розробки для керування процесом розробки (разом з робочими документами). У всякому разі, вони повинні бути перевірені (протестовані) на відповідність програмам ПЗ. Ці документи утворять два комплекти з різним призначенням:

1. Користувальницька документація ПЗ.
2. Документація по супроводу ПЗ.

2.1 Документація користувача

Вона необхідна, якщо програмний засіб припускає яку-небудь взаємодію з користувачами.

Користувальницька документація ПЗ *(user documentation)* пояснює користувачам, як вони повинні діяти, щоб застосовувати дане ПЗ.

Ці документи частково торкають питання супроводу ПЗ, але не стосуються питань, пов'язаних з модифікацією програм.

До користувальницької документації відносяться документи, якими керується користувач:

- при інсталяції ПЗ (при установці ПЗ із відповідним настроюванням на сферу застосування);
- при застосуванні ПЗ для рішення своїх завдань;

- при керуванні ПЗ (наприклад, коли дане ПЗ взаємодіє з іншими системами).

У зв'язку із цим варто розрізняти дві категорії користувачів ПЗ:

1. Кінцеві користувачі ПЗ – *end-user*.
2. Адміністратори ПЗ – *system administrator*.

Кінцевий користувач використовує ПЗ для рішення своїх завдань (у своїй предметній області). Це може бути інженер, що проектує технічний пристрій, або касир, що продає залізничні квитки за допомогою ПЗ. Він може й не знати багатьох деталей роботи комп'ютера або принципів програмування.

Адміністратор управляє використанням ПЗ кінцевими користувачами й здійснює супровід ПЗ, не пов'язане з модифікацією програм. Наприклад, він може регулювати права доступу доД ПЗ між кінцевими користувачами, підтримувати зв'язок з постачальниками ПЗ або виконувати певні дії, щоб підтримувати ПЗ у робочому стані, якщо це включено як частина робот ПЗ у іншу систему.

Склад користувальницької документації залежить від категорії користувачів, на які орієнтоване дане ПЗ, і від режиму використання документів. Під категорією тут розуміється контингент користувачів ПЗ, у якого є необхідність у певній користувальницькій документації ПЗ. Вдалий користувальницький документ істотно залежить від точного визначення категорії, для якої він призначений. Користувальницька документація повинна містити інформацію, необхідну для кожної категорії. Під режимом використання документа розуміється спосіб, що визначає, яким образом використається цей документ. Звичайно користувачеві досить великих програмних систем потрібні або документи для вивчення ПЗ (використання у вигляді інструкції), або для уточнення деякої інформації (використання у вигляді довідника).

Можна вважати типовим наступний склад користувальницької документації для досить великих ПЗ:

1. Загальний функціональний опис ПЗ. Дає коротку характеристику функціональних можливостей ПЗ. Призначено для користувачів, які повинні вирішити, наскільки необхідно їм дане ПЗ.
2. Посібник з інсталяції ПЗ. Призначено для системних адміністраторів. Цей документ повинен детально пропонувати, як установлювати систему в конкретному середовищі. Він повинен

містити опис носіїв, на якому поставляється ПЗ, що представляють ПЗ файли, а також вимоги до мінімальної конфігурації апаратур.

3. Інструкція із застосування ПЗ. Документ призначений для кінцевих користувачів. Містить необхідну інформацію із застосувань ПЗ, що організована в зручній для вивчення формі.

4. Довідник по застосуванню ПЗ. Цей документ призначений для кінцевих користувачів. Містить необхідну інформацію із застосувань ПЗ, організовану в зручній для виборчого пошуку окремих деталей формі.

5. Посібник з керування ПЗ. Призначено для системних адміністраторів. Воно повинне описувати повідомлення, що генеруються, коли ПЗ взаємодіє з іншими системами, і як реагувати на ці повідомлення. Крім того, якщо ПЗ використає системні апаратури, цей документ може пояснювати, як супроводжувати ці апаратури.

Як уже говорилося раніше, розробка користувальницької документації починається відразу після створення зовнішнього опису. Якість цієї документації може істотно визначати успіх ПЗ. Вона повинна бути досить проста й зручна для користувача (у противному випадку це ПЗ взагалі не коштувало створювати). Тому, хоча чорнові варіанти (начерки) користувальницьких документів створюються основними розроблювачами ПЗ, до створення їхніх остаточних варіантів часто залучаються професійні технічні письменники. Крім того, для забезпечення якості користувальницької документації розроблений ряд стандартів, у яких пропонується порядок розробки цієї документації, формулюються вимоги до кожного виду користувальницьких документів і визначається їхня структура й зміст.

2.2 Документація по супроводу

Ця документація необхідна, якщо ПЗ припускає вивчення того, як воно влаштовано (сконструйоване), і модернізацію його програм.

Документація по супроводу ПЗ *(system documentation)* описує ПЗ із погляду його розробки.

Як уже відзначалося, супровід - це триваюча розробка. Тому якщо буде потреба модернізації ПЗ до цієї роботи залучається спеціальна група розроблювачів-супровідників. Цій групі доведеться

мати справу з документацією, що визначала діяльність проектної групи основних розроблювачів ПЗ, - з тією лише різницею, що ця документація для розроблювачів-супровідників буде “чужою” (вона створювалася іншими людьми). Група розроблювачів-супровідників повинна буде вивчати цю документацію:

- для того, щоб зрозуміти будову й процес розробки модернизируемого ПЗ;
- внести в цю документацію необхідні зміни, повторюючи в значній мірі технологічні процеси, за допомогою яких створювалося початкове ПЗ.

Документацію по супроводу ПЗ можна розбити на дві групи:

1. Документація, що визначає будову програм і структур даних ПЗ і технологію їхньої розробки.
2. Документація, що допомагає вносити зміни в ПЗ.

Документація першої групи містить підсумкові документи кожного технологічного етапу розробки ПЗ. Вона включає наступні документи:

1. Зовнішній опис ПЗ *(Requirements document)*.
2. Опис архітектури ПЗ *(Description of the system architecture)*, включаючи зовнішню специфікацію кожної її програми.
3. Для кожної програми ПЗ - опис її модульної структури, включаючи зовнішню специфікацію кожного включеного в неї модуля.
4. Для кожного модуля – його специфікація й опис його будови *(Design description)*.
5. Тексти модулів обраною мовою програмування *(Program source code listings)*.
6. Документи встановлення достовірності ПЗ *(Validation documents)*, що описують, як встановлювалася достовірність кожної програми ПЗ і як інформація про встановлення достовірності зв'язувалася з вимогами до ПЗ. Ці документи включають насамперед документацію по тестуванню (схема тестування й опис комплекту тестів), але можуть включати й результати інших видів перевірки ПЗ, наприклад, докази властивостей програм.

Документація другої групи містить:

Посібник із супроводу ПЗ *(system maintenance guide)*, що

- описує відомі проблеми в ПЗ;
- описує, які частини системи є апаратно- і программно-залежними;

- описує, як розвиток ПЗ прийнятий у розрахунок при його конструюванні.

Загальна проблема супроводу ПЗ - забезпечити, щоб всі його подання залишалися узгодженими, коли ПЗ змінюється. Щоб цьому допомогти, зв'язки та залежності між документами і їхніми частинами повинні бути зафіксовані в базі даних управління конфігурацією.

3 Автоматизація розробки програмного забезпечення

При розробці складних програмних продуктів групою учасників важливо вирішувати такі питання, як:

1. Організація взаємодії розроблювачів, що створюють систему.
2. Здійснення контролю над версіями системи.
3. Організація безпечного зберігання компонент, що створюються в процесі роботи над системою.
4. Організація повторного використання компонент.
5. Забезпечення інтеграції інструментів, використаних розроблювачами.

Для розробки ПЗ використовуються моделі. Для побудови моделей застосовуються візуальні засоби моделювання, які дозволяють:

1. Виконати декомпозицію ПЗ, розбивши її на складові частини (підсистеми) з метою більш зручного розуміння людиною.
2. Описати систему в наочній для людини графічній формі із застосуванням стандартної нотації (наприклад, *UML*).
3. Показати різні аспекти ПЗ (наприклад, функціонування – *Use case View*, логічну організацію – *Logical View*, фізичну структуру – *Component View, Deployment View*).
4. Знайти необхідні рішення, що відносяться до програмної реалізації (наприклад, сценарії взаємодії об'єктів *Sequence Diagrams*, паралельні потоки *Activity Diagrams*, потоки даних *Collaboration Diagrams*).
5. Спроектувати базу даних.
6. Виявити повторно використані програмні компоненти, що надасть можливість скоротити обсяг кодування.

7. Виконувати побудову моделей із програмного коду й бази даних при проектуванні успадкованих систем (*Reverse engineering*).

Заснований на моделях підхід застосовується як в об'єктно-орієнтованной технології, так і в структурній. Для кожної із цих технологій визначен свій набір інструментальних засобів.

Однак, створення ПЗ - це не тільки побудова моделі. Чим складніше й ширше проект, тим більше він вимагає інструментальної підтримки всіх етапів життєвого циклу програмного забезпечення.

3.1 Особливості й компоненти case-засобів

Computer-Aided Software Engineering – система автоматизованої розробки програм, або *CASE-засіб*.

Звичайно до *CASE-засобів* відносять будь-який програмний засіб, що автоматизує один процес або сукупність процесів життєвого циклу програмного забезпечення.

Сучасні *CASE-засоби* мають наступні характерні риси:

1. Наявність графічних засобів для опису й документування систем, що забезпечують зручний інтерфейс із розроблювачем і розвиваючих його творчі можливості.
2. Інтеграція окремих компонентів, що забезпечує керованість процесом розробки систем.
3. Використання спеціальним образом організованого сховища (репозиторія) проектних метаданих (артефактів).

Комплекс засобів, що підтримують повний життевий цикл(ЖЦ) ПЗ (інтегрований *CASE-засіб*), містить наступні компоненти:

1. Репозиторій, що забезпечує:
   - зберігання версій проекту і його окремих частин;
   - синхронізацію надходження інформації від різних розробників при колективній розробці;
   - контроль даних на повноту й несуперечність.
2. Графічні засоби аналізу й проектування, які забезпечують створення й редагування моделей (діаграм, сценаріїв) ПЗ.
3. Засоби розробки додатків, включаючи системи програмування й керування базами даних (генерацію вихідних кодів по моделях на різних мовах програмування; генерацію схем баз

даних для Систем Керування Базами Даних (СКБД); зв'язок між засобом проектування, системою програмування та СКБД).

4. Засоби конфігураційного керування. Конфігураційним керуванням називається діяльність по систематичному обліку та контролю внесення обгрунтованих змін у програмний продукт. Система конфігураційного керування дає можливість:

- відстежити, які існують об'єкти в проекті,
- у якому стані вони перебувають,
- як завантажені виконавці,
- як виконуються завдання й т.п.

5. Засоби документування.
6. Засоби тестування.
7. Засоби керування проектом.
8. Засоби реінжинірингу (трансформації успадкованого ПЗ у нове ПЗ).

За використовуваною технологією створення систем можна класифікувати *CASE-засоби* на об’єктно-оріентовані й структурні. Основними компонентами тих чи інших засобів автоматизації є засоби аналізу й проектування.

3.2 Об’єктно-оріентовані case-засоби аналізу та проектування

Світовим лідером засобів аналізу й проектування об’єктно-оріентованих систем є продукт *Rational Rose* фірми *IBM Rational Software* (США). Цей *CASE-засіб* призначений для автоматизації етапів аналізу й проектування.

Робота в *Rational Rose* полягає в проектуванні певного виду діаграм, задаючи при цьому всі властивості, зв'язки та взаємодію елементів моделі один з одним.

При розробці будь-якої системи виникає проблема взаєморозуміння виконавця й замовника. Маючи такий інструмент, як *Rose*, аналітик завжди може показати замовникові не абстрактний словесний опис процесу, а його конкретну модель (на екрані персонального комп'ютера чи в друкованому виді – неважливо). *Rose* дозволить швидше погодити із замовником всі деталі планованої системи. Результатом моделювання є файл із моделлю, що аналітик передає наступній ланці співробітників - програмістам, які доповнюють отриману логічну модель системи моделями конкретних класів для конкретної мови програмування.

Для моделювання об’єктно-оріентований засіб *Rose* використовує:

1. Уніфіковану мову моделювання *(Unified Modeling Language – UML)*.
2. Об'єктну модель програмних компонентів *(Component Object Model – COM)*.
3. Техніку об'єктного моделювання *(Object Modeling Technique – OMT)*.
4. Метод візуального моделювання Г. Буча' 93 *(Booch'93)*.

Багатий набір можливостей *Rose* надає розроблювачам:

1. Проектування систем з кодогенерацією. Дозволяє перетворити модель на опис конкретною мовою програмування. Підтримуються мови: *C++, Ada, Java, Basic, XML (eXtensible Markup Language), Oracle*. Також до *Rose* сторонніми компаніями розробляються спеціальні мости до не вхідящих у стандартну поставку мов, наприклад, до *Delphi*.
2. Обернене проектування (реінжиніринг), коли готову систему (наприклад, на *C++*) або базу даних (на *Oracle*) “закачують” в *Rose* з метою одержання наочної візуальної (структурної) моделі.
3. Зворотнє проектування *(Round-trip engineering)*, що об'єднує можливості перших двох підходів, коли створюється система, а по проходженні деякого часу еволюційного періоду (доробок) піддається знову реінжинірингу та кодогенерації.

Тепер *Rational Rose* поставляється в наступних редакціях:

1. *Rose DataModeler* – дозволяє проектувати системи й бази даних без можливості кодогенерації. Продукт спрямований на архітекторів і аналітиків.
2. *Rose RealTime* – узкоспеціалізована версія для систем реального часу, здатна проводити 100% кодогенерацію та реінжиніринг тільки для мов *C и C++*. Має неповний набір діаграм. Продукт спрямований лише на програмістів.
3. *Rose Enterprise* – найбільш повна версія, містить у собі всі вищеописані можливості. Продукт спрямований на архітекторів, аналітиків, програмістів.

3.3 Структурні case-засоби

Сутність структурного підходу до розробки ПЗ полягає в її декомпозиції на функції що автоматизуються: система розбивається від функціональних підсистем до конкретних процедур.

Принципи об'єктно-орієнтованого підходу: "розділяй і пануй" (розбивка складних проблем на безліч простих для розуміння завдань) і ієрархічне впорядкування (організація складових частин системи в деревоподібні структури з додаванням нових деталей на кожному рівні) - зберігаються так само й при структурному підході. Однак при цьому підході аналізуються не об'єкти і їх взаємодія, а алгоритми виконаних системою функцій і відносини між даними при виконанні цих функцій.

Структурний підхід найчастіше використовується при аналізі й проектуванні баз даних.

Майже всі *CASE-засоби* структурного аналізу й проектування систем застосовують наступні моделі:

1. *Data Flow Diagrams (DFD)* – діаграми потоків даних спільно зі словниками даних і специфікаціями процесів.
2. *Entity-Relationship Diagrams (ERD)* – діаграми "сутність-зв'язок".
3. *State Transition Diagrams (STD)* – діаграми переходів станів.

Логічна діаграма потоків даних *(DFD)* показує зовнішні стосовно системи джерела й приймачі даних. Структури потоків даних і визначення їхніх компонентів зберігаються в словнику даних.

Діаграма "сутність-зв'язок" *(ERD)* розкриває модель сховища даних і забезпечує стандартний спосіб визначення даних і відносин між ними. Тут ідентифікуються важливі для даної предметної області сутності (об'єкти), властивості цих сутностей (атрибути) і їхні відносини з іншими сутностями (асоціації).

Діаграма переходів станів *(STD)* є засобом опису поведінки системи, що залежить від реального часу. При цьому враховується початковий стан (стартова крапка для системного переходу); перехід (переміщення системи з одного стану в інше); умова (подія або події, що викликають перехід); дія (операція, що може мати місце при виконанні переходу).

Відомим *CASE-засобом* структурного аналізу й проектування є *Silverrun* – засіб американської фірми *Silverrun Technologies Inc.*

Silverrun складається із чотирьох модулів, кожний з яких є самостійним продуктом і може використовуватись без зв'язку з іншими модулями:

1. *Business Process Modeler (BPM)* будує моделі у вигляді діаграм потоків даних.
2. *Entity-Relationship eXpert (ERX)* забезпечує побудову моделей даних “сутність-зв'язок”.
3. *Relational Data Modeler (RDM)* дозволяє створювати деталізовані моделі “сутність-зв'язок”, призначені для реалізації в реляційній базі даних.
4. *Workgroup Repository Manager (WRM)* застосовується як словник даних для зберігання загальної для всіх моделей інформації.

Для автоматичної генерації схем баз даних в *Silverrun* існують мости до найпоширеніших СКБД: *Oracle, DB2, SQL Server, MS Access*. Крім того, є програмні мости до об’єктно-орієнтованого *CASE-засобу Rational Rose*, розроблені російською фірмою «Аргуссофт».

Багатьма розроблювачами інформаційних систем використовуються *CASE-засоби BPWin* і *ERWin* фірми *Platinum technology*, які призначені для аналізу, проектування й кодогенерації. Фірма *Platinum* має програмні мости з *Rational Rose* для зв'язування моделі даних з об'єктною моделлю. У книгарнях є книга: *Маклаков С. В. BPWin і ERWin. CASE-засоби розробки інформаційних систем. – М.: ДИАЛОГ-МИФИ, 1999.* Книга містить опис методів структурного аналізу й проектування моделей даних. Докладно на конкретних прикладах розглянуте застосування *CASE-технологій* і *CASE-засобів* для автоматизації етапів аналізу, проектування й кодогенерації інформаційних систем.

3.4 Case-засоби компанії ibm rational software

Процес створення програмного забезпечення доволі складний. Певні незручності розроблювачам доставляють вимоги користувачів до системи, що часто змінюються, операційні системи, що розростаються не по днях (а по годинах). Розроблювачам доводиться, у буквальному значенні слова “тримати ніс за вітром”, аналізуючи всі віяння, переводячи їх у машинний код. У результаті код програми розростається, штат розроблювачів теж, а час на

випуск кожного нового релізу скорочується. Тим самим складається дуже цікава ситуація, коли випуск ПЗ не встигає не тільки за вимогами користувачів, але й за виходом чергових апаратних і програмних новинок. Якщо ж продукт встиг вийти вчасно, то, як правило, містить багато помилок, оскільки не всі компанії тримають величезний штат кваліфікованих тестерів. А якщо до всього перерахованого вище ще додати зростаючу інтернаціоналізацію, коли команди розроблювачів розкидані по всій планеті! Об'єктно-орієнтований підхід і мови програмування, маючи багатьма достоїнства *(conceptual integrity, contract, selfishness, hierarchy, seamlesness, encapsulation, inheritance, polymorphism)*, самі по собі всеодно не можуть вирішити проблему швидкісної розробки.

По статистиці (дослідження компанії *Standish Group CHAOS*) тільки 26% проектів закінчуються успішно (читай: "вчасно"), тобто тільки чверть усього задуманого втілюється в життя. Аналіз діяльності найбільш успішних компаній і аналіз їхніх секретів успіху дозволив створити загальну рекомендацію для всіх інших. Компанія *IBM Rational Software* постійно займається дослідженнями в області Інформаційних Технологій з метою вироблення оптимального шляху в створенні програмного забезпечення. Девіз компанії: *"Будуй швидше, надійніше, якісніше…"*Основа всього, що випускає компанія – універсальний процес *(RUP – Rational Unified Process)*.

3.4.1 Засіб візуального моделювання Rational Rose

Rose – засіб візуального моделювання бізнесів-процесів, аналізу вимог і проектування на основі трирівневої архітектури компонентів. *CASE-засіб Rational Rose* є продуктом №1 у програмному списку *IBM Rational Software* (див.лек. 9.2).

Перший етап життєвого циклу повністю підтримується *Rose*.

Наступним кроком на шляху побудови грамотного ПЗ є побудова документообігу у відповідності з усіма можливими стандартами, щоб документи цілком і повністю відбивали реальний стан справ. Інструментом автоматизації документообігу займається наступний продукт *Rational SoDA*.

3.4.2 Автоматизоване документування Rational SoDA

Результатом майже будь-якої діяльності є документ або звіт заздалегідь установленого зразка. Всю роботу з написання може взяти на себе технічний письменник, але йому буде дуже важко “виймати” з моделей специфікації й коментарі, переносячи в текстовий редактор, – це неправильний підхід. На рішення всіх проблем з документообігом спрямований інструмент *Rational SoDA.*

Основний обов'язок *Rational SoDA* – підготувати звіт по заздалегідь установленому шаблоні.

Дані для звіту можуть бути взяті з будь-якого продукту *Rational*. Наприклад, готовий документ по наявній в *Rose* моделі. Набір звітів, підтриманих *SoDA*, такий: *Rational Rose, Requisite PRO, ClearCase* і *TeamTest.*

SoDA є макросом для *MS Word*. Система викликів (і меню) інтегрована в *MS Word* і дозволяє генерувати шаблони на базі наявних файлів. *SoDA* допускає використання як стандартних шаблонів, так і створених користувачем за допомогою спеціального майстра *(Wizard)*, також убудованого в систему меню *MS Word.*

Продукт обов'язковий для використання в складі будь-якого проекту. Також особливу спрямованість продукт має на розроблювачів і постановників завдань.

Розглянуті три програмних продукти *(RUP, Rose, SoDA)* повністю покривають потреби архітекторів, аналітиків, програмістів і технічних письменників. Правда, вирвані з контексту процесу розробки, вони не забезпечують повноцінної підтримки життєвого циклу розробки. Для цього універсальний процес рекомендує використання наступних продуктів: *RUP, SoDA, Requisite PRO* і *ClearQuest.*

ПЗ будь-яких розмірів неможливо будувати у відриві від консультацій із замовником. Подібні консультації ведуть до постійного уточнення функцій системи (зміні вимог). Якщо користуватися обє’ктно-орієнтованим підходом і інструментальним засобом *Rational Rose*, то (з погляду компанії) 90% функцій буде обговорено на самому початку розробки. Отже, всі зміни не будуть носити шоковий характер. *IBM Rational Software* пропонує набір інструментальних засобів для того, щоб перетворити внесення нових вимог у повсякденний безболісний процес.

3.4.3 Управління вимогами Rational RequisitePRO

RequisitePRO – це інструмент, що підтримує оновлення й відслідковує зміни у вимогах для всього колективу розроблювачів, представляючи їх у зручному виді для читання, обговорення й змін.

Для кожної вимоги:

- підтримується розширюваний набір атрибутів, що дозволяє характеризувати вимоги відповідно до уявлень користувача;
- призначається конкретний виконавець;
- зберігається його історія, що дозволяє відстежити, які зміни були внесені у вимогу, ким, коли й чому;
- встановлюється пріоритет, по якому вимоги сортуються.

RequisitePRO дозволяє:

1. Централізовано *організувати всі документи й дані*, що стосуються вимог: вимоги замовника, дизайн підсистем, сценарії, структурні й функціональні специфікації, плани тестування.
2. Візуально *визначати схожі вимоги* в рамках одного або декількох проектів. Це дає можливість застосування готових апробованих рішень у новому проекті.
3. *Задати зв'язки між вимогами*. Це дозволяє легко простежити, які вимоги варто піддати аналізу (і, можливо, перегляду) при модифікації деякої конкретної вимоги або атрибута. Тим самим спрощується процес внесення змін.
4. *Спростити спілкування між розроблювачами* шляхом надання спільного доступу до всіх вимог проекту, або до їхньої частини.

Продукт направлений на всіх учасників проекту.

3.4.4 Керування запитами на зміни Rational ClearQuest

Через постійні зміни в проекті життєво необхідно знати статистику того, що змінювалося, ким і в який час, причому мати повний обсяг необхідної інформації в якій-небудь базі даних (наприклад, під керуванням СКБД *Oracle*).

ClearQuest – *Windows*- і *Web*- розміщуваний продукт, що допомагає колективу розроблювачів відслідковувати й управляти

всіма діями по зміні програмного забезпечення протягом усього життєвого циклу.

Запити на зміни проходять цикл із декількох станів *(States)*:

1. Тільки що поданий запит перебуває в стані “Поданий” *(Submitted)*.
2. Після передачі запиту співробітникові він переходить у стан “Призначений” *(Assigned)*.
3. Початок роботи над запитом переводить його в стан “Відкритий” *(Open)*, і вся команда може бачити, що хтось обробляє запит.
4. Коли запит перевіряється, він проходить стадію “Перевірка” *(Verify)*.
5. Коли запит закритий, він переходить у стан “Закритий” *(Resolved)*.

Таким чином, будь-який учасник проекту (від підлеглого до керівника) може користуватися базою у власних цілях (дивитися й/або складати власні запити).

Даний продукт через *World Wide Web* підтримує зв'язок усередині команд, розділених територіально.

Продукт направлений на всіх учасників команди.

Отже, розглянуті п'ять програмних продуктів, чотири з яких необхідні протягом усього циклу розробки програмної системи *(RUP, ClearQuest, RequisitePRO, SoDA)* і один, необхідний на самих відповідальних етапах проектування й кодогенерації *(Rose)*. Всі перераховані вище продукти основні, але тільки ними в процесі розробки не обійтися.

Для одержання якісного коду розроблювач повинен позбавити код помилок і настроїти його на максимальну продуктивність, потім передати код тестерам для виявлення логічних (високорівневих) помилок.

Наступні три програми допоможуть розроблювачеві істотно підвищити якість коду, знизивши при цьому час, необхідний на детальне пророблення коду.

3.4.5 Вимірювання характеристик Rational Quantify

Quantify – засіб для збору й аналізу інформації про продуктивність створеного програмного продукту, що дозволяє виміряти характеристики додатка і його компонентів.

Засоб орієнтований на розроблювачів, що програмують на *C/C++, Basic* і *Java*, і допомагає визначати й усувати вузькі місця в продуктивності програмного забезпечення.

Даний продукт:

1. Генерує в табличній формі список всіх функцій, що викликаються у процесі роботи додатка, указуючи тимчасові характеристики кожної з них.
2. Надає повну статистику по всіх викликах (зовнішніх і внутрішніх), незважаючи на розміри тестуємого додатка й час його тестування. Збір даних здійснюється за допомогою технології *OCI (Object Code Insertion)*. Суть способу складається в підрахунку всіх машинних циклів шляхом вставки лічильників у код для кожного функціонального блоку. Всі цикли додатка прораховуються фактично, а не за допомогою довільних вибірок, як у більшості пакетів тестування. Унікальність даного підходу полягає в тім, що тестуються всі використовувані компоненти (наприклад, бібліотеки *DLL*), а також у тім, що для аналізу зовсім необов'язково мати вихідні тексти тестуємого додатка.
3. Запам'ятовує всі попередні виклики й дає їх порівняльну оцінку. Наприклад, при багаторазових перекомпіляціях додатка.
4. Має можливість перенести в *Microsoft Excel* статистичну інформацію з викликів, де можна побудувати як графіки, так і зведені таблиці для різних запусків програми.

Продукт орієнтований на розроблювачів.

3.4.6 Пошук помилок виконання Rational Purify

Purify – це засіб пошуку помилок виконання *(run-time)* для додатків і компонентів мов *C/C++* і вирішення проблем, пов'язаних з втратами пам'яті.

Багато програмних продуктів замикають на себе під час роботи всі системні ресурси без великої на те необхідності, що може привести готову систему до краху в самий відповідальний момент. Виникнення подібного роду помилок досить важко відстежити стандартними засобами.

Загалом робота *Purify* зводиться до висновку детальнішої статистики про використання пам'яті додатком. Програма збирає дані про будь-які втрати в пам'яті. Наприклад, неповернення блоку, не

використання покажчиків, зупинка виконання програми з виводом стану середовища при виникненні помилки *run-time*.

Purify надає можливість не тільки бачити попередження й помилки, але й переходити до відповідного вихідного тексту. Така можливість існує тільки для внутрішніх викликів, оскільки вихідні тексти динамічних бібліотек поки недоступні.

Продукт орієнтований на розроблювачів.

3.4.7 Області коду, що не піддаються тестуванню Rational PureCoverage

PureCoverage – засіб, основним і єдиним призначенням якого є виявлення ділянок коду, пропущених при тестуванні додатка.

Розроблювачі можуть урахувати це й більш ретельно виконувати перевірку.

Провести тестування абсолютно всіх функцій розроблювач не може, тому що неможливо на 100% бути впевненим у тім, що всі протестовано.

PureCoverage збирає статистику про ті ділянки програми, які під час тестування не були пройдені (виконані). Подібні рядки підсвічуються червоним кольором, чітко вказуючи на наявність чорних дір у програмі у вигляді не протестованого коду. Додатково програма лічить так називані активно виконувані рядки - хіти.

Отже, розроблювач може оцінити, скільки раз виконався кожний рядок, що складає функцію. Маючи подібну статистику, простіше виявити рядки, що не виконались, і проаналізувати причину, по якій вони не одержали управління.

Продукт спрямований на розроблювачів.

Від тестування коду програми переходять до функціонального тестування. Це наступний етап у розробці ПЗ, коли визначені всі вимоги до неї, коли написана якась працездатна версія програми або окремого сегмента.

Наступні три основні програми тестування спрямовані на високорівневе тестування. Кожна із програм не просто дозволяє щось тестувати, а здатна створювати спеціальні скрипти (сценарії) для наступного повторного тестування.

3.4.8 Візуальне тестування Rational Visual Test

Visual Test є автоматизованим інструментом тестування, що забезпечує надійне функціональне тестування.

Visual Test має інтерфейс, подібний з *Visual Studio* від компанії *Microsoft*. З його допомогою можна тестувати:

- 32-бітний *Windows-додаток*;
- компонент *Active*,
- *DLL-бібліотеку*;
- *Web-додаток*.

Основу *Visual Test* становить похідна від *Visual Basic* розширена мова програмування *Test Basic*, із сотнями специфічних для тесту функцій, спеціальних конструкцій для полегшення тестування й відкритою архітектурою, що робить цю мову розширюваною.

Продукт орієнтований на тестерів і розроблювачів.

3.4.9 Тестування користувальницького інтерфейсу Rational Robot

Robot – інструментальний засіб функціонального тестування, що базується на об'єктно-орієнтованій технології, призначений для тестування графічного інтерфейсу (*GUI-тестування*).

Програма здатна працювати у двох режимах: ручному й автоматичному.

У ручному режимі користувач сам задає спеціальною мовою сценарій тестування (скрипт).

В автоматичному режимі користувач тестує додаток, а *Robot* автоматично генерує необхідний скрипт для подальшого повторного тестування. Скрипти можна редагувати, відладжувати і настроювати.

Продукт орієнтований на тестерів і розроблювачів.

3.4.10 Тестування розподілених додатків Rational LoadTest

LoadTest – засіб автоматизованого тестування характеристик розподілених мережевих додатків на платформах *Windows* і *Unix*.

При цьому тестуванні типово використовується навантаження сервера більшою кількістю віртуальних користувачів. Наприклад, можна встановити таймер для одного користувача, щоб визначити яка кількість часу займе виконання запиту, коли тисячі

інших користувачів посилають запити на той же самий сервер у той же самий час.

Таким чином, стає можливим при використанні даної програми перевіряти на продуктивність систему архітектури *C/S* (клієнт/сервер).

Продукт спрямований на тестерів, розроблювачів, *Web-розроблювачів*.

Залишилося розглянути тільки сполучну ланку - програмний продукт, який би використовувався на всіх етапах проекту: від ідеї і до впровадження.

3.4.11 Конфігураційний і версійний контроль Rational ClearCase

ClearCase є засобом конфігураційного й версійного контролю, що зберігає всю історію кожного файлу проекту з метою можливого порівняння змін (включаючи міграцію файлів між незалежними проектами) і дозволяє отримувати останні версії файлів, що редагують, протягом усього життєвого циклу.

Даний продукт:

1. Керує робочим простором.
2. Забезпечує єдине середовище, інструментальні засоби й підхід до роботи.
3. Забезпечує доступ кожному учаснику проекту, як до всіх файлів проекту, так і тільки до певної його частини. Для досягнення подібного ефекту *ClearCase* використовує систему фільтрів, що настроюються та приховують непотрібну інформацію.
4. Дозволяє здійснити паралельну розробку.
5. Дозволяє окремому учаснику проекта виходити із загального состава розробки, забираючи роботу “додому”, а після всіх внесених змін повертати версії знову в проект. При цьому *ClearCase* здійснить автоматичне злиття версій.
6. Дозволяє поєднувати географічно віддалені команди розроблювачів за допомогою *MultiSite* (спеціального модуля, що здійснює передачу поточного стану проекту на зазначений сайт).
7. Має механізми інтеграції з іншими засобами розробки (наприклад, *Oracle*), з різними додатковими генераторами звітів (наприклад, *MS Word*) та ін.

Продукт орієнтований на всіх учасників команди: керівників, менеджерів, розроблювачів, аналітиків, тестерів, технічних письменників.

Питання для самоконтролю

1. Особливості у розробці великих програмних проектів.
2. Основні етапи повного циклу розробки програмного продукту.
3. Які завдання вирішуються на етапі дослідження проблеми?
4. Що враховується при визначенні економічної доцільності розробки програмного продукту?
5. Що потрібно робити при визначенні структури системи, що проектується?
6. Чи можливо не лише інтуїтивне визначення якості програмної системи?
7. Чи можливо керувати якістю програмного продукту?
8. Що та які спеціалісти роблять під час безпосереднього програмування компонентів?
9. Як і ким здійснюється інтеграція окремих компонентів в єдиний продукт?
10. Які завдання вирішуються під час тестування програмних продуктів?
11. Охарактеризуйте основні джерела помилок в ході розробки системи.
12. Які вам відомі методи тестування систем?
13. Для чого використовують засоби документування результатів тестування?
14. Що таке Rational Rose і які основні ознаки цього продукту?
15. Як поєднані між собою Rational Rose і мова моделювання UML?
16. Навіщо потрібні інтегральні середовища розробки?
17. Яка повинна бути документація до програмних продуктів?
18. Як організується взаємодія між розробниками та замовниками при створенні програмних продуктів?
19. Які ви можете навести рекомендації щодо підбору складу коллективу розробників програмної системи?
20. Як правильно розподіляти роботу між виконавцями згідно з компонентною структурою системи?
21. Як контролювати виконання етапів робіт?
22. Яка звітність рекомендована у процесі розробки программного проекту і як її використовувати?

Лабораторна робота 1.

Тема: Організація розробки програмного забезпечення

Мета: Формулювання мети розробки ПЗ та організація колективу розробників.
Тривалість: 4 години
Завдання: Самостійно обирається напрямок та тема програмного продукту. Формується група проекту та розподіляються ролі учасників групи. Організовується модель проектної групи.
Очікуваний результат: звіт, який містить

1) Мету та задачі розробки програмного продукту.
2) Опис предметної області….
3) Перелік учасників групи
4) Ролі кожного учасника (з попереднім описом обов'язків)

Порядок виконання:

1) Розподіл на групи.
2) Обговорення напряму розробки ПЗ та формулювання цілей та задач розробки.
3) Попередній розподіл ролей та обов'язків учасників групи.
4) Створення звіту

Теоретичні відомості: див. п.1
(Визначення термінів «Ціль розробки», «Задачі розробки», типовий склад групи розробників, типові ролі, приклади)
Додаткова література: див. список літератури: основна [1, 2,4,10,16], додаткова [21,24]

Лабораторна робота 2

Тема: Розробка технічного завдання

Мета: набуття практичних навичок по складанню технічного завдання на розробку програмного забезпечення.
Тривалість: 8 годин
Завдання:

1) Описати типи користувачів ПЗ (навички, мета роботи, вік, кваліфікація …)
2) Описати ВСІ варіанти використання ПЗ (які операції виконує кожен користувач, як часто, які навички для цього потрібні)
3) Структура всіх класів та зв'язків між ними (CASE моделювання, UML)
4) Вибір засобів реалізації
5) Послідовність розробки.
6) Розподіл обов'язків між учасниками групи.

Очікуваний результат: звіт, який містить попереднє технічне завдання

Порядок виконання:

1) Описати модель користувача ПЗ (навички, мета роботи, вік, кваліфікація …)
    a. Ознайомитися з прикладами типи користувачів;
    b. Вибрати, найбільш підходящі типи.
2) Описати ВСІ варіанти використання ПЗ (які операції виконує кожен користувач, як часто, які навички для цього потрібні)

Заповнити таблицю (кількість рядків залежить від напряму розробки)

№	Короткий опис дій користувача	Типи користувачів	Частота виконання (часто, середня частота, зрідка, у виключних випадках)

3) Схема інтерфейса
    Схема кожного вікна та діалога програмного продукту. Наприклад для додатка з графічним інтерфейсом користувача потрібно
    a. Спираючись на варіанти використання, скласти перелік вікон та діалогів,
    b. Показати приблизний зовнішній вигляд кожного вікна
    c. Показати переходи між вікнами.
4) Структура всіх класів та зв'язків між ними (CASE моделювання, UML)

a. Заповнити CRC картки для кожного класу [http://en.wikipedia.org/wiki/Class-Responsibility-Collaboration_card]

   Назва класу: _____
   Суперкласи та підкласи:_____
   _____
   Функції класу: _____
   _____
   _____
   _____
   З якими классами зв'язаний: _____
   _____
   _____

b. Вибрати програмний засіб моделювання;
c. За допомогою обраного засобу моделювання створити діаграму класів

5) Для кожного варіанту використання виписати класи, які потрібні для його реалізації, створити діаграму кооперації класів.
6) Вибрати засоби реалізації (мова програмування, середовище розробки та ін.)
7) Опираючись на залежності між класами (див. попередній крок), визначити послідовність розробки та кількість розробників, розподілити обов’язки.
8) Створення звіту

Теоретичні відомості: див. п.3
Додаткова література: див. список літератури: основна [2, 5, 6, 9, 11, 19], додаткова [22,28]

Лабораторна робота 3

Тема: Колективна розробка та тестування програм за технічним завданням

Мета: набуття практичних навичок по розробці та тестуванню програмного забезпечення з використанням засобів колективної розробки.

Тривалість: 16 годин

Завдання: На базі розробленого технічного завдання розробити та протестувати ПЗ.

Очікуваний результат: звіт, який містить

1) Визначити стиль програмування.
2) Очікуваний результат працюючого програмного коду.
3) Аналіз якості отриманого програмного коду.
4) Обробку можливих помилок
5) Автоматично створену документацію.
6) Опис тестів (які дані використовувались при тестуванні, програмні засоби та методики тестування, отриманий результат, зручність та легкість використання розробленого ПЗ).
7) Конфігураційний и версій ний контроль

Порядок виконання:

1) Вибрати систему контролю версій, або обґрунтувати її відсутність.
2) Створити правила форматування коду та іменування змінних, функцій та ін. (стиль програмування).
3) Розробка програмного інтерфейсу функцій та класів і його паралельне документування;
4) Розробити код модулів програмного продукту.
    a. кодування реалізацій;
    b. обробка можливих помилок.
    c. розробка тестів для кожного модуля програми.
    d. оновлення версій продукту (після реалізації кожного модуля його вихідні коди відправляються іншим розробникам).
5) Інтеграція модулів (збірка модулів в єдиний програмний продукт, тестування продукту як цілісної системи).
6) В залежності від можливостей середовища розробки автоматично створити документацію.
7) Опис тестування розробленого програмного продукту:
    a. опис даних, які використовувались при тестуванні;
    b. програмні засоби та методики тестування;
    c. отриманий результат;

d. оцінка використання розробленого ПЗ (зручність, легкість у використанні,..).

Теоретичні відомості: див. п.3

Додаткова література: див. список літератури: основна [3, 7, 8, 12, 13, 14, 18,19, 20], додаткова [26, 27]

Лабораторна робота 4

Тема: Документація процесу розробки програмного забезпечення

Мета: набуття практичних навичок по розробці комплекту супровідних документів з розробки програмного забезпечення.

Тривалість: 6 години

Завдання: Документація процесу розробки програмного забезпечення

Очікуваний результат: звіт, який містить

1) Документи керування розробкою ПЗ
   a) Плани, оцінки, розклади(створюються менеджерами).
   b) Звіти використання ресурсів у процесі розробки (створюються менеджерами).
   c) Стандарти.
   d) Робочі документи.
   e) Замітки та переписка.
2) Документи, що входять до складу ПЗ
   a) Користувальницька документація ПЗ.
   b) Документація по супроводу ПЗ.
   c) Документація, що допомагає вносити зміни в ПЗ:

Порядок виконання:

1. Документи керування розробкою ПЗ
   a. Плани, оцінки, розклади(створюються менеджерами).
   b. Звіти використання ресурсів у процесі розробки (створюються менеджерами).
   c. Стандарти.
   d. Робочі документи.
   e. Замітки й переписка.

2. Документи, що входять до складу ПЗ
3. Користувальницька документація ПЗ.
    a. Загальний функціональний опис ПЗ.
    b. Посібник з інсталяції ПЗ.
    c. Інструкція із застосування ПЗ.
    d. Довідник по застосуванню ПЗ.
    e. Посібник з керування ПЗ.
4. Документація по супроводу ПЗ.
    a. Документація, що визначає будову програм і структур даних ПЗ і технологію їхньої розробки:
    b. Зовнішній опис ПЗ.
    c. Опис архітектури ПЗ, включаючи зовнішню специфікацію кожної її програми.
    d. Для кожної програми ПЗ - опис її модульної структури, включаючи зовнішню специфікацію кожного включеного в неї модуля.
    e. Для кожного модуля – його специфікація й опис його будови.
    f. Тексти модулів обраною мовою програмування.
    g. Документи встановлення достовірності ПЗ
        a. документація по тестуванню (схема тестування й опис комплекту тестів)
        b. результати інших видів перевірки ПЗ, наприклад, доказу властивостей програм.
5. Документація, що допомагає вносити зміни в ПЗ (посібник із супроводу ПЗ):
    a. описує відомі проблеми в ПЗ;
    b. описує, які частини системи є апаратно- і программно-залежними;
    c. описує, як розвиток ПЗ прийнят у розрахунок при його конструюванні.

Теоретичні відомості: див. п.2

Додаткова література: див. список літератури: основна [7, 13, 15, 17, 20], додаткова [25]

ЛІТЕРАТУРА

Основна

1. Боггс У. UML и Rational Rose. / Боггс У., Боггс М. / Пер. с англ. – М.: ЛОРИ, 2008, – 580с.
2. Боэм Б. Инженерное проектирование программного обеспечения. / Пер. с англ. – М.: Радио и связь, 1985, – 512с.
3. Боэм Б., Браун Дж., Каспар Х. и др. Характеристики качества программного обеспечения. / Пер. с англ. – М.: Мир, 1981, – 208с.
4. Брукс Ф. Мифический человеко-месяц или как создаются программные системы./ Пер. с англ. – СПб.: Символ-Плюс, 2001, – 304с.
5. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++./ Пер. с англ. – 2-е изд. – М.: Бином, СПб.: Невский диалект, 2001, – 560с.
6. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя. / Пер. с англ. – М.: ДМК, 2000.
7. Вендров А. М. CASE-технологии. Современные методы и средства проектирования информационных систем. [Електронний ресурс]: – М.: Финансы и статистика, 1998. – Режим доступу: http://www.citforum.ru/database/case/index.shtml
8. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приёмы объектно-ориентированного проектирования. Паттерны проектирования. / Пер. с англ. – СПб.: Питер, 2007, – 366с.
9. Гома Х. UML. Проектирование систем реального времени, параллельных и распределённых приложений. / Пер. с англ. – М.: ДМК Пресс, 2002, – 704с.
10. Зиглер К. Методы проектирования программных систем. / Пер. с англ. – М.: Мир, 1985, – 328с.
11. Йордан Э., Аргила К. Структурные модели в объектно-ориентированном анализе и проектировании. / Пер. с англ. – М.: ЛОРИ, 1999, – 288с.
12. Калбертсон Р., Браун К, Кобб Г. Быстрое тестирование. / Пер. с англ. – М.: Вильямс, 2002, – 384с.
13. Калянов Г. Н. CASE структурный системный анализ. – М.: ЛОРИ, 1996, – 250с.

14. Канер С. Тестирование программного обеспечения. / Канер С., Фолк Д., Кек Нгуен Е / Пер. с англ. – Киев.: ДиаСофт, 2001, – 544с.

15. Кантор М. Управление программными проектами. Практическое руководство по разработке успешного программного обеспечения. / М.: Вильямс, 2002. - 176 с.

16. Кармайкл Э. Быстрая и качественная разработка программного обеспечения. / Кармайкл Э., Хейвуд Д. / Пер. с англ. – М.: Вильямс, 2003, – 400с.

17. Кватрани Т. Rational Rose 2000 и UML. Визуальное моделирование. / Пер. с англ. – М.: ДМК, 2001, – 176с.

18. Керниган Б. В., Пайк Р. Практика программирования. / Керниган Б. В., Пайк Р. / Пер. с англ. – М.: Вильямс, 2004, – 288с.

19. Коналлен Д. Разработка Web-приложений с использованием UML. / Пер. с англ. – М.: Вильямс, 2001, – 288с.

20. Крачтен Ф. Введение в Rational Unified Process. / Пер. с англ. – 2-е изд. – М.: Вильямс, 2002, – 240с.

Додаткова

21. ISO/IEC 12207:1995, Information Technology — Software life cycle processes, 1995. Amendments 2002, 2004

22. ГОСТ Р-1999. ИТ. Процессы жизненного цикла программных средств, 2000

23. ISO/IEC 15288:2002, Systems engineering — System life cycle processes, 2002

24. ISO/IEC 15504-1-9, Information technology — Process assessment, Parts 1-9. 15504-1,3,4:2004, 15504-2:2003/Cor 1:2004, TR 15504-5:2004

25. IEEE 1074-1997 IEEE Standard for Developing Software Life Cycle Processes, 1996

26. А. Якобсон Унифицированный процесс разработки программного обеспечения. / А. Якобсон, Г. Буч, Дж. Рамбо. – СПб.: Питер, 2002, – 496с.

27. Л. Басс Архитектура программного обеспечения на практике. / Л. Басс, П. Клементс, Р. Кацман. – СПб.: Питер, 2006, – 576с.

28. Г. Буч Язык UML. Руководство пользователя. / Г. Буч, Дж. Рамбо, А. Джекобсон. – М.: ДМК Пресс, 2003, – 432с.

ЗМІСТ

ВСТУП 3
ВИМОГИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ 5
1 ОРГАНІЗАЦІЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 6
1.1 Група проекту та ролі учасників групи 6
1.1.1 Керівник проекту - Project manager 6
1.1.2 Менеджер по маркетингу - Product manager 7
1.1.3 Розроблювач - Developer 7
1.1.4 Тестер - Tester 8
1.1.5 Технічний письменник - Writer 8
1.1.6 Представник групи технічної підтримки - Logistic 9
1.1.7 Інші фахівці 9
1.2 Модель проектної групи 9
2 ДОКУМЕНТАЦІЯ ПРОЦЕСУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 12
2.1 Документація користувача 13
2.2 Документація по супроводу 15
3 АВТОМАТИЗАЦІЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 17
3.1 Особливості й компоненти CASE-засобів 18
3.2 Об’єктно-орієнтовані CASE-засоби аналізу та проектування 19
3.3 Структурні CASE-засоби 21
3.4 CASE-засоби компанії IBM Rational Software 22
3.4.1 Засіб візуального моделювання Rational Rose 23
3.4.2 Автоматизоване документування Rational SoDA 24
3.4.3 Управління вимогами Rational RequisitePRO 25
3.4.4 Керування запитами на зміни Rational ClearQuest 25
3.4.5 Вимірювання характеристик Rational Quantify 26
3.4.6 Пошук помилок виконання Rational Purify 27
3.4.7 Області коду, що не піддаються тестуванню Rational PureCoverage 28
3.4.8 Візуальне тестування Rational Visual Test 29
3.4.9 Тестування користувальницького інтерфейсу Rational Robot 29
3.4.10 Тестування розподілених додатків Rational LoadTest ... 29
3.4.11 Конфігураційний і версійний контроль Rational ClearCase 30

Питання для самоконтролю.....32
Лабораторна робота 1.....33
Лабораторна робота 2.....33
Лабораторна робота 3.....35
Лабораторна робота 4.....37
Література.....39

Методичне видання
(українською мовою)

Тодоріко Ольга Олексіївна

ТЕХНОЛОГІЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Методичні вказівки
для студентів II курсу математичного факультету

Рецензент В.А. Єрмолаєв

Відповідальний за випуск С.Ю. Борю

Коректор Г.А. Добровольський