

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Запорізький національний технічний університет

**ПРОГРАМА, МЕТОДИЧНІ ВКАЗІВКИ
ТА КОНТРОЛЬНІ ЗАВДАННЯ**
з дисципліни
**“ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ
СИСТЕМ”**

для студентів напрямку 123 ”Комп'ютерна інженерія”
спеціалізацій (освітніх програм) ”Комп'ютерні системи та мережі”,
”Спеціалізовані комп'ютерні системи”
заочної форми навчання

2017 р.

Програма, методичні вказівки та контрольні завдання з дисципліни "Програмне забезпечення комп'ютерних систем" для студентів напряму 123 "Комп'ютерна інженерія" спеціалізацій (освітніх програм) "Комп'ютерні системи та мережі", "Спеціалізовані комп'ютерні системи" заочної форми навчання/ Укл.: К.М. Касьян, М.М. Касьян. - Запоріжжя: ЗНТУ, 2017. - 56 с.

Укладачі: К.М. Касьян, доцент, к.т.н.
М.М. Касьян, доцент, к.т.н.

Рецензент М.Ю.Тягунова, доцент, к.т.н.

Відповідальний
за випуск: К.М. Касьян, доцент, к.т.н.

Затверджено
на засіданні кафедри КСМ
Протокол № 4 від 14.11.2017

Затверджено
на засіданні НМК КНТ
Протокол № 4 від 28.11.2017

ЗМІСТ

ЗАГАЛЬНІ МЕТОДИЧНІ ВКАЗІВКИ	4
1 РОБОЧА ПРОГРАМА І МЕТОДИЧНІ ВКАЗІВКИ ЩОДО ВИВЧЕННЯ ТЕМ ДИСЦИПЛІНИ	5
1.1 Призначення й класифікація програмного забезпечення	5
1.2 Вимоги до розробки ПЗ. Визначення програмного засобу, комплексу програм, програми	9
1.3 Життєвий цикл програмних засобів (ПЗ), його основні тапи. Особливості ПЗ із малою та великою тривалістю життєвого циклу	12
1.4 Стадії розробки ПЗ	15
1.5 Специфікація ПЗ. Методи розробки специфікацій ПЗ	18
1.6 Документування ПЗ. Види програмних документів	22
1.7 Проектування модульної структури ПЗ. Основні структури модульних програм.....	24
1.8 Структурування програмних компонентів. Висхідне та спадне проектування ПЗ. Елементарні базові структури	27
1.9 Надійність функціонування комплексів програм. Фактори, що знижують надійність функціонування програм. Програмні методи підвищення надійності функціонування КП	32
1.10 Супровід програмного забезпечення	37
1.11 Реінжиніринг програмного забезпечення.....	41
1.12 Автоматизація підтримки життєвого циклу ПЗ.....	45
2 Перелік лабораторних робіт	48
3 КОНТРОЛЬНІ ЗАВДАННЯ	48
3.1 Загальні вказівки.....	48
3.2 Завдання на контрольну роботу	48
3.3 Методичні вказівки до виконання контрольної роботи.....	50
Література.....	56

ЗАГАЛЬНІ МЕТОДИЧНІ ВКАЗІВКИ

Вивчення дисципліни полягає в засвоєнні студентами основних принципів і методів технології проектування програмних систем (ПС) на всіх стадіях їх, життєвого циклу. Студенти повинні ознайомитись з основними стадіями розробки ПС, що в сукупності відображають повний цикл її проектування. Кожна стадія завершується розробкою проектної та експлуатаційної документації.

Дисципліна "Програмне забезпечення комп'ютерних систем" вивчається студентами заочної форми навчання в одинадцятому семестрі. Основна форма вивчення дисципліни - самостійна проробка матеріалу за літературою, вказаною по розділах цих методичних указівок.

Увесь матеріал дисципліни складається з розділів. На початку кожного розділу дається відповідна частина програми дисципліни, потім коротко викладаються основні положення, на які повинна бути звернена увага при вивченні теоретичних питань даного розділу по рекомендованій літературі. Для самоконтролю ступеня засвоєння проробленого матеріалу студент повинен користуватися запитаннями для самоперевірки, що подаються в кожному розділі вказівок.

У процесі роботи над дисципліною студент зобов'язаний виконати одну контрольну роботу та ряд лабораторних робіт. По кожній виконаній лабораторній роботі студент повинен скласти звіт та захистити її перед викладачем.

Успішне засвоєння дисципліни "Програмне забезпечення комп'ютерних систем" можливе тільки при систематичній роботі над підручниками. Матеріал, що вивчається рекомендується конспектувати.

1 РОБОЧА ПРОГРАМА І МЕТОДИЧНІ ВКАЗІВКИ ЩОДО ВИВЧЕННЯ ТЕМ ДИСЦИПЛІНИ

1.1 Призначення й класифікація програмного забезпечення

Програмне забезпечення (ПЗ). Призначення ПЗ. Системне та прикладне ПЗ. Операційні системи та операційні оболонки. Системні утиліти. Засоби контролю. Системи програмування. Пакети прикладних програм. Прикладні програми користувачів. [1(с. 231-258), 3(с. 9-13), 4(161-175)].

1.1.1 Методичні вказівки

Під програмним забезпеченням (ПЗ) розуміється сукупність програм, процедур і правил разом із зв'язаною з цими компонентами документацією, що дозволяє використовувати комп'ютерну систему (КС) для рішення різних задач.

ПЗ дозволяє удосконалити організацію роботи обчислювальної системи з метою максимального використання її можливостей, підвищити продуктивність і якість праці користувачів, адаптувати програми користувача до ресурсів конкретної обчислювальної системи, розширити ПЗ обчислювальної системи.

У залежності від функцій, виконуваних різними елементами ПЗ, його можна розділити на дві групи: системне (загальне) та прикладне (спеціальне). Класифікація програмного забезпечення приведена на рисунку 1.1. До системного програмного забезпечення відносяться програми, описи та інструкції, призначені для автоматизації трудомістких технологічних етапів розробки алгоритмів і програм, для організації та контролю обчислювального процесу, а також для керування розподілом ресурсів під час функціонування обчислювальної системи.

Прикладне програмне забезпечення являє собою сукупність програм рішення задач із різних сфер людської діяльності.

Системне ПЗ за характером використання окремих елементів розділяється на операційні системи, операційні оболонки, системні утиліти, засоби контролю та системи програмування.



Рисунок 1.1 – Класифікація програмного забезпечення

Операційна система (ОС) являє собою частину ПЗ, що призначена для керування процесом обробки програм користувачів від моменту їхнього надходження в систему до видачі результатів, а також для розподілу ресурсів комп'ютерної системи між окремими програмами та користувачами.

Операційна оболонка (ОО) являє собою програму-надбудову до операційної системи, що забезпечує доступ користувача до команд і ресурсів ОС за допомогою більш зручного інтерфейсу та реалізуючу додаткові функції розподілу ресурсів обчислювальної системи і керування файлами.

Системні утиліти – це програми, що розширюють можливості операційних систем і операційних оболонок у частині підключення нових периферійних пристроїв, кодування інформації та керування ресурсами комп'ютера.

Засоби контролю – сукупність програмно-апаратних засобів КС для виявлення помилок у процесі роботи комп'ютера. Вони призначені для перевірки працездатності окремих вузлів, блоків та всієї машини в

цілому. Засоби контролю можна розділити на засоби діагностики КС, програмно-логічний контроль КС, тестовий контроль, апаратний контроль та програмно-апаратний контроль.

Засоби діагностики забезпечують автоматичний пошук помилки та виявлення несправностей з визначеною локалізацією їх в КС та її окремих модулях.

Програмно-логічний контроль, заснований на використанні надлишкового коду вихідних та проміжних даних КС (додатковий розряд при контролі на парність чи непарність, код Хеммінга), що дозволяє знаходити помилки при зміні значення окремих бітів даних.

Тестовий контроль здійснюється за допомогою спеціальних тестів для перевірки правильності роботи КС чи її частин.

Апаратний контроль здійснюється автоматично, за допомогою вбудованого в КС устаткування.

Програмно-апаратний контроль включає програмний та апаратний контроль.

Системи програмування містять програмні засоби, призначені для реалізації операційних алгоритмів (програм) у вигляді деякої послідовності розпізнаваних комп'ютером інструкцій (кодів, команд). Система програмування включає систему команд процесора, периферійних пристроїв, транслятори, компілятори та інтерпретатори різних мов програмування. По ступені залежності вихідного програмного коду від конкретної обчислювальної системи розрізняють машинні коди, машинно-залежні та машинно-незалежні засоби програмування. При цьому під засобом програмування розуміється як алгоритмічна мова, що визначає синтаксис набору інструкцій, так і програма перетворення цих інструкцій у машинні коди.

До машинно-залежних програмних засобів відносяться мови типу Асемблера, в основу яких покладена машинно-орієнтована алгоритмічна мова.

Машинно-незалежні програмні засоби будуються на основі алгоритмічних мов високого рівня і призначені для автоматизації процесу програмування, зниження трудомісткості розробки програм та підвищення їхньої надійності.

Прикладне програмне забезпечення складається з пакетів прикладних програм (ППП) та прикладних програм користувачів.

Значне місце в прикладному ПЗ займають пакети прикладних програм, що по сфері застосування поділяються на проблемно-орієнтовані та інтегровані пакети.

Відмітною рисою проблемно-орієнтованих ППП є їх порівняно вузька спрямованість на визначене коло розв'язуваних задач і велика їхня розмаїтість. Інтегровані ППП містять у собі інструментальні засоби, кожне з яких по функціональних, обчислювальних можливостях практично рівносильна проблемно-орієнтованій програмі. У структурі цих пакетів передбачений модуль керування, що забезпечує переключення між додатками і безконфліктне використання загальних даних.

ППП є могутнім інструментом автоматизації розв'язуваних користувачем задач, практично цілком звільняючи його від необхідності знати, як виконує машина ті чи інші процедури по перетворенню інформації. Саме ППП служать найважливішим джерелом розвитку ПЗ сучасних обчислювальних систем, прискорюють і полегшують впровадження обчислювальної техніки до різних сфер діяльності.

Прикладні програми створюються користувачем з використанням засобів програмування, що мають у його розпорядженні в складі конкретної обчислювальної системи. У цьому випадку створення та налагодження програм здійснюються кожним користувачем індивідуально відповідно до правил і угод того ППП чи КС, у рамках яких вони застосовуються.

1.1.2 Запитання для самоперевірки

1.1.2.1 Що розуміється під програмним забезпеченням?

1.1.2.2 В чому полягає призначення ПЗ?

1.1.2.3 Як класифікується ПЗ?

1.1.2.4 Призначення операційних систем.

1.1.2.5 Призначення операційних оболонок.

1.1.2.6 Призначення системних утиліт.

1.1.2.7 На які три види можна поділити засоби контролю та у чому їхнє призначення?

1.1.2.8 На які три види можна поділити системи програмування та у чому їхнє призначення?

1.1.2.9 В чому відмінність пакетів прикладних програм та прикладних програм користувачів?

1.1.2.10 В чому відмінність проблемно-орієнтованих та інтегрованих пакетів прикладних програм?

1.2 Вимоги до розробки ПЗ. Визначення програмного засобу, комплексу програм, програми

Модульна структура. Прирощуваність і розвиток. Надійність. Передбачуваність. Зручність і ергономічність. Гнучкість. Ефективність. Сумісність. Принцип параметричної універсальності. Принцип функціональної надмірності. Принцип функціональної вибірковості. Програмний засіб. Комплекс програм. Програма. [1(с. 9-12), 4(с. 166-168)].

1.2.1 Методичні вказівки

ПЗ орієнтовано на використання комп'ютерних систем (КС) у різних сферах діяльності і повинне забезпечувати своєчасне та адекватне поставленим задачам рішення.

Це викликає необхідність дотримання ряду вимог при розробці компонентів ПЗ, основними з яких є: модульність, природуваність і розвиток, надійність, передбачуваність, зручність і ергономічність, гнучкість, ефективність, сумісність.

Для реалізації можливостей, наданих користувачу, необхідно, щоб програмна система мала модульну структуру. Розбивка великої системи на окремі, що піддаються огляду та аналізу частини спрощує розробку, але вимагає чіткої організації робіт.

Вимога природуваності і розвитку визначаються двома обставинами: по-перше, сам користувач має потребу в змінах, а, по-друге, ніяке ПЗ не є статичним. Розроблене ПЗ можна використовувати як базову систему для побудови більш могутніх та досконалих систем. При цьому витрати на її розвиток значно зменшуються.

Вимога надійності припускає, що розроблювальне ПЗ підвищує надійність функціонування обчислювальної системи при рішенні тих задач, для яких воно призначено. ПЗ повинно визначати та діагностувати помилки, а також самовідновлюватися після більшості характерних помилок користувача. Воно повинно захищати свої програми, програми користувача і дані від помилок, принаймні

зводити до мінімуму шкоду, що вони можуть зробити для всього програмного оточення.

Вимога передбачуваності означає, що ПЗ повинно реагувати на дії користувачів передбачуваним чином. Час, що необхідний для виконання тих самих операцій, не повинне варіюватися занадто сильно. Результат виконання команд при дотриманні тих самих умов і на тих самих даних не повинний розрізнятися.

Вимога зручності і ергономічності ПЗ припускає наявність дружнього та інтуїтивна зрозумілого інтерфейсу, необхідність обліку основних фізіологічних та психофізіологічних факторів діяльності людини при створенні ПЗ.

Вимога гнучкості означає можливість налаштування ПЗ на різні умови функціонування та класи однотипних задач.

Вимога ефективності зв'язана з можливістю за допомогою ПЗ вчасно і точно вирішувати поставлені задачі при оптимальному використанні ресурсів обчислювальної системи та праці користувача. У ряді випадків для кількісної оцінки ефективності ПЗ використовується співвідношення "продуктивність/вартість": чим воно вище, тим ПЗ ефективніше.

Вимога сумісності означає можливість застосування ПЗ, а також використовуваних їм та формованих даних в іншому обчислювальному середовищі (на іншому комп'ютері, лініях передачі даних, в інших операційних середовищах і т.д.). Сумісність досягається за рахунок наявності стандартної системи команд і форматів даних, стандартизації процедур вводу-виводу інформації.

В основу розробки сучасного ПЗ покладені три основних принципи - параметрична універсальність, функціональна надмірність, функціональна вибірковість.

Принцип параметричної універсальності полягає в орієнтації ПЗ на різні апаратні конфігурації машин. Навіть програмно-сумісні КС мають різний кількісний та якісний склад периферійних пристроїв, різний обсяг оперативної та зовнішньої пам'яті, а також інших ресурсів.

Функціональна надмірність має на увазі задоволення декількох суперечливих вимог користувачів. На практиці часто зустрічається випадок, коли виграш у часі роботи програми приводить до різкого росту обсягів займаною програмою пам'яті. Щоб мати можливість у рамках одного програмного засобу заощаджувати або час, або пам'ять,

створюються дві чи більш програми, що реалізують ту саму функцію, але, економлячи визначений обчислювальний ресурс. Цей принцип знайшов широке застосування при розробці КС.

Функціональна вибірковість означає включення до складу ПЗ мінімально необхідних можливостей. Додаткові функціональні можливості, розраховані на різні сфери застосування та конфігурацію машин, поставляються користувачу за його вимогою.

Програмний засіб (ПЗ) – сукупність програм визначеного призначення, придатних для виконання на КС, що пройшла випробування з зафіксованими показниками якості та постачених комплектом документації, достатньої для кваліфікованої експлуатації по призначенню та використання як продукції виробничо-технічного призначення.

Комплекс програм (КП) – сукупність взаємозалежних програм для КС, в основному як об'єкт розробки на різних етапах його створення, однак ще, що не досяг завершеного стану, придатного для тиражування й експлуатації з визначеними якісними показниками.

Тому в процесі аналізу технології проектування переважно використовується термін "комплекс програм" і тільки після успішного завершення випробувань – термін "програмний засіб".

У поняття програма включаються тексти будь-яких програм мовою програмування чи в об'єктному коді, придатному для виконання на КС.

Програмні засоби реалізуються на різних типах КС, характеристики яких визначаються призначенням і складністю ПС. В комп'ютерних системах у якості реалізуючих комп'ютерів можуть застосовуватися універсальні великі та комп'ютери загального призначення, спеціалізовані і бортові комп'ютери, а також персональні комп'ютери. Загальні принципи проектування ПЗ для КС досить універсальні, і основні особливості проектування зв'язані з призначенням ПЗ, з доступними ресурсами застосовуваних КС, з відповідністю їхньому призначенню і складності розв'язуваних програмами задач.

Ці фактори впливають на раціональний рівень автоматизації проектування, на розмір і складність взаємодії в колективі розроблювачів, на трудомісткість і тривалість створення ПЗ і т.д. Однак принципи та методи проектування ПЗ при цьому змінюються відносно мало.

1.2.2 Запитання для самоперевірки

- 1.2.2.1 Назвіть основні вимоги, яких необхідно дотримуватись при розробці компонентів ПЗ.
- 1.2.2.2 Для чого потрібна модульна структура ПЗ?
- 1.2.2.3 Чим обумовлена вимога прирощуваності і розвитку ПЗ?
- 1.2.2.4 В чому полягає вимога надійності ПЗ?
- 1.2.2.5 Що означає вимога передбачуваності ПЗ?
- 1.2.2.6 В чому полягає вимога зручності і ергономічності ПЗ?
- 1.2.2.7 Що означає вимога гнучкості ПЗ?
- 1.2.2.8 В чому полягає вимога ефективності ПЗ?
- 1.2.2.9 Що означає вимога сумісності ПЗ?
- 1.2.2.10 В чому полягає принцип параметричної універсальності ПЗ?
- 1.2.2.11 В чому полягає принцип функціональної надмірності ПЗ?
- 1.2.2.12 В чому полягає принцип функціональної вибірковості ПЗ?
- 1.2.2.13 Дайте визначення програмному засобу.
- 1.2.2.14 Що таке комплекс програм?
- 1.2.2.15 Що означає поняття "програма"?

1.3 Життєвий цикл програмних засобів (ПЗ), його основні тапи. Особливості ПЗ із малою та великою тривалістю життєвого циклу

Життєвий цикл програмних засобів. Програми з малою тривалістю експлуатації. Програми з великою тривалістю експлуатації. Системний аналіз. Проектування ПЗ. Експлуатація ПЗ. Супровід ПЗ. [1(с. 22-24), 3(13-16)].

1.3.1 Методичні вказівки

Життєвий цикл ПЗ містить у собі всі етапи розвитку від виникнення потреби в програмі визначеного цільового призначення до повного припинення використання цього ПЗ внаслідок його морального старіння чи втрати необхідності рішення відповідних задач.

По тривалості життєвого циклу ПЗ можна розділити на два класи: з малим та великим часом життя. Програми з малою тривалістю експлуатації створюються в основному для рішення наукових та інженерних задач, для одержання конкретних результатів обчислень. Такі програми звичайно відносно невеликі (1-10 тис. команд), розробляються одним фахівцем чи маленькою групою, не призначені для тиражування й передачі для наступного використання в інші колективи. Такі програми є частиною науково-дослідної роботи і не можуть розглядатися як відчужувані вироби та програмні засоби. Їхній життєвий цикл складається з тривалого інтервалу системного аналізу та формалізації проблеми, значного етапу проектування програм та невеликого часу експлуатації й одержання результатів. Вимоги до функціональних і конструктивних характеристик, як правило, не формалізуються, відсутні оформлені випробування програм, та показники їхні якості контролюються тільки розроблювачами відповідно до неформальних уявлень. Супровід і модифікація таких програм не потрібні, та їхній життєвий цикл завершується після одержання результатів обчислень. Основні витрати в життєвому циклі таких програм приходяться на етапи системного аналізу та проектування, що продовжуються від місяця до 1-2 років, у результаті чого життєвий цикл рідко перевищує 3 роки.

Програми з великою тривалістю експлуатації створюються для регулярної обробки інформації та керування в процесі функціонування складних ОС. Розміри ПЗ можуть змінюватися в широких межах (1-1000 тис. команд), однак усі вони мають можливість модифікації в процесі тривалого супроводу та використання різними фахівцями. Програми цього класу допускають тиражування, вони супроводжуються документацією як промислові вироби і являють собою відчужуваний програмний продукт. Проектуванням та експлуатацією ПЗ можуть займатися великі колективи фахівців, для чого необхідна формалізація необхідних технічних характеристик КП та їхніх компонентів, а також формалізовані випробування та визначення досягнутих показників якості ПЗ. Життєвий цикл таких ПЗ складає 10-20 років, з яких 70-90% приходить на експлуатацію та супровід. Внаслідок масового тиражування і тривалого супроводу сукупні витрати в процесі експлуатації та супроводу можуть значно перевищити витрати на системний аналіз та проектування.

Далі будемо розглядати великі програмні комплекси систем керування та обробки інформації, оформлені як програмний продукт. Життєвий цикл розглянутих програм містить у собі наступні основні етапи (рисунок 1.2):

- системний аналіз, у ході якого визначаються потреба в ПЗ, його призначення та основні функціональні характеристики, оцінюються витрати і можлива ефективність застосування такого комплексу програм;
- проектування ПЗ, що включає в себе розробку структури комплексу та його компонент, програмування модулів та ряд етапів налагодження, а також випробування і впровадження для регулярної експлуатації створеної версії КП;
- експлуатацію ПЗ, що полягає у функціонуванні програм на ЕОМ для обробки інформації й одержання результатів, які є метою створення ПЗ, а також у забезпеченні вірогідності та надійності виданих даних;
- супровід ПЗ, що складається в експлуатаційному обслуговуванні, розвитку функціональних можливостей та підвищенні експлуатаційних характеристик ПЗ, у тиражуванні та переносі КП на різні типи обчислювальних засобів.

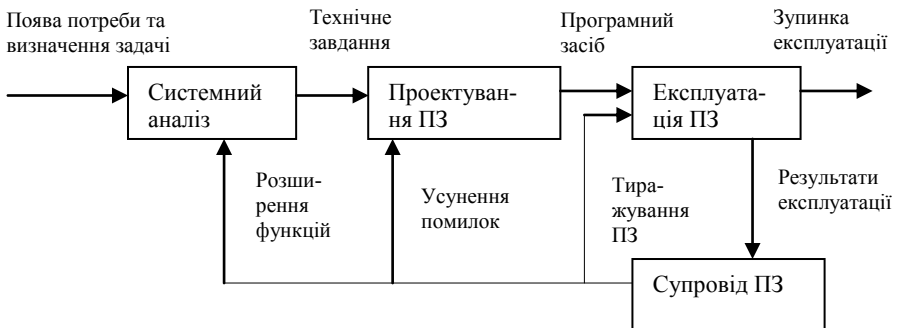


Рисунок 1.2 - Життєвий цикл програмних засобів

Серед перерахованих етапів найбільш специфічним та тісно зв'язаним з функціональним призначенням ПЗ є етап системного аналізу. На цьому етапі формулюються призначення та основні

показники якості створюваних програм. Розв'язувані задачі практично цілком визначаються предметною областю системного аналізу. Тому на даному етапі важко узагальнювати технологічні процеси та критерії якості при створенні різних типів програм.

Етапи проектування, експлуатації та супроводу значно розрізняються цілями, задачами, методами та засобами. При цьому ПЗ відповідно розглядається як об'єкт розробки, чи як функціонуючий виріб, чи як об'єкт контролю та модифікації.

Супровід відіграє роль необхідного зворотного зв'язку від етапу експлуатації. У процесі функціонування програм можливе виявлення помилок у програмах та з'являється необхідність їхньої модифікації й розширення функцій. Ці доробки, як правило, ведуться одночасно з експлуатацією поточної версії ПЗ. Після перевірки підготовлених коректувань чергова версія замінює ті, що раніше експлуатувалися чи деякі з них. При цьому процес експлуатації може бути практично беззупинним тому, що заміна версій є короткочасною. Це приводить до того, що процес експлуатації версії ПЗ йде паралельно та незалежно від етапу супроводу.

1.3.2 Запитання для самоперевірки:

1.3.2.1 Що таке життєвий цикл ПЗ?

1.3.2.2 В чому полягають особливості програм з малою тривалістю життєвого циклу?

1.3.2.3 В чому полягають особливості програм з великою тривалістю життєвого циклу?

1.3.2.4 Які задачі вирішуються на етапі системного аналізу?

1.3.2.5 Що містить у собі етап проектування ПЗ?

1.3.2.6 В чому полягає етап експлуатації ПЗ?

1.3.2.7 В чому складається етап супроводу ПЗ?

1.4 Стадії розробки ПЗ

Технічне завдання. Ескізний проект. Зовнішнє проектування. Технічний проект. Проектування архітектури. Проектування структур даних. Проектування модульної структури. Проектування модулів. Робочий проект. Стадія впровадження. [2(с. 10-12)].

1.4.1 Методичні вказівки

Державний стандарт встановлює наступні стадії розробки: технічне завдання, ескізний, технічний та робочий проекти, упровадження.

Технічне завдання (ТЗ). На цій стадії виконуються наступні види робіт: постановка задачі; збір вихідних матеріалів; вибір та обґрунтування критеріїв ефективності і якості розроблювальної програми; обґрунтування необхідності проведення науково-дослідних робіт; доцільності застосування раніше розроблених програм, принципової можливості рішення поставленої задачі; попередній вибір методів рішення задач; визначення вимог до технічних засобів; вимог та цілей розробки програм; стадій, етапів і термінів розробки програми та документації на неї; розробка техніко-економічного обґрунтування розробки програми; узгодження та затвердження ТЗ.

При розробці програмного виробу пред'являються вимоги до функцій, що повинні забезпечуватися виробом; часовим та об'ємним характеристикам програм; до сумісності з іншими програмами; надійності і вірогідності; до методів і засобів настроювання на конкретні умови експлуатації; до зручності використання та супроводу.

Ескізний проект (ЕП). На цій стадії виконуються наступні види робіт:

- зовнішнє проектування програмного виробу;
- уточнення методів рішення задачі;
- попереднє проектування внутрішніх структур даних;
- розробка загального алгоритму рішення задачі, укрупненої структурної схеми програмного виробу, пояснювальної записки.

Зовнішнє проектування програмного виробу являє собою процес опису очікуваного поведіння системи з погляду користувача. Ціль даного процесу – проектування зовнішньої взаємодії користувача з програмним виробом.

При розробці ЕП визначаються: способи взаємодії користувачів із програмним виробом; функції користувачів; тип мови взаємодії; синтаксис і семантика мови взаємодії; структура й зміст інформаційних кадрів та шаблонів діалогу; структура вхідних і вихідних даних.

Технічний проект (ТП). Етапи технічного проектування: проектування архітектури програмного виробу; структур даних;

модульної структури виробу; модулів; розробка пояснювальної записки.

На етапі проектування архітектури програмний виріб розбивають на складові частини; визначають функції кожного компонента, способи взаємодії між ними; розробляють схеми керування потоками даних, схеми розподілу оперативної та зовнішньої пам'яті обчислювальної системи.

На етапі проектування структур даних визначають способи представлення, збереження та перетворення вхідних, вихідних і внутрішніх даних.

На етапі проектування модульної структури здійснюється розбивка компонентів програмного виробу на модулі (процедури), визначення їхніх функцій та способів взаємодії.

На етапі проектування модулів складається опис усіх модулів програмного виробу, що містить у собі: ім'я модуля; функції, виконувані модулем; опис вхідних і вихідних параметрів, їхніх форматів, діапазонів припустимих значень і т.д.; функціонального взаємозв'язку між вхідними та вихідними параметрами; подій, зовнішніх стосовно модуля (друк повідомлень, робота з файлами і т.д.); форму звертання до модуля; список модулів, що викликають даний, то модулів, що викликає даний модуль; опис алгоритму модуля.

Робочий проект (РП). На цій стадії виконуються наступні види робіт: кодування, тестування та налагодження програм; розробка програмних документів відповідно до вимог ЄСПД; проведення різних видів приймально-здавальних випробувань; коректування програм і документації за результатами випробувань.

Стадія впровадження. На цій стадії здійснюється підготовка та передача програм і програмної документації для супроводу та виготовлення.

Таким чином, процес розробки ПЗ складається з наступних основних етапів: визначення вимог, проектування, кодування, перевірка та документування. Етап проектування є найважливішим етапом створення ПЗ, на якому закладаються основні характеристики програм та визначається зміст робіт на інших етапах.

1.4.2 Запитання для самоперевірки

1.4.2.1 Перелічите стадії розробки ПЗ.

1.4.2.2 Які види робіт виконуються на стадії технічного завдання?

1.4.2.3 Які види робіт виконуються на стадії ескізного проекту?

1.4.2.4 Перелічите етапи технічного проектування.

1.4.2.5 Які види робіт виконуються на етапі проектування архітектури?

1.4.2.6 Які види робіт виконуються на етапі проектування структур даних?

1.4.2.7 Які види робіт виконуються на етапі проектування модульної структури?

1.4.2.8 Які види робіт виконуються на етапі проектування модулів?

1.4.2.9 Які види робіт виконуються на стадії робочого проекту?

1.4.2.10 Які види робіт виконуються на стадії впровадження?

1.5 Специфікація ПЗ. Методи розробки специфікацій ПЗ

Специфікація ПЗ. Етапи процесу розробки вимог. Вимоги користувача. Системні вимоги. Проектна системна специфікація. Функціональна специфікація. Нефункціональна специфікація. [2(с. 12-23)].

1.5.1 Методичні вказівки

Процес формування специфікації, призначений для визначення сервісів, якими володітиме проєктоване ПО, а також обмежень, що накладаються на функціональні можливості і розробку програмної системи. Цей процес в даний час звичайно називають "розробка вимог" (requirements engineering). Розробка вимог часто є критичним етапом в створенні ПО, оскільки помилки, допущені на цьому етапі, ведуть до виникнення проблем на етапах проектування і розробки.

Схема процесу розробки вимог показана на рис. 1.3. Результатом його виконання є розробка документації, що формалізує вимоги, що пред'являються до системи, тобто створення системної специфікації. В цій документації вимоги звичайно представлені на двох рівнях деталізації. На самому верхньому рівні представлені вимоги, визначені кінцевими користувачами або замовниками ПЗ; але для розробників необхідна системна специфікація, що більш деталізується.

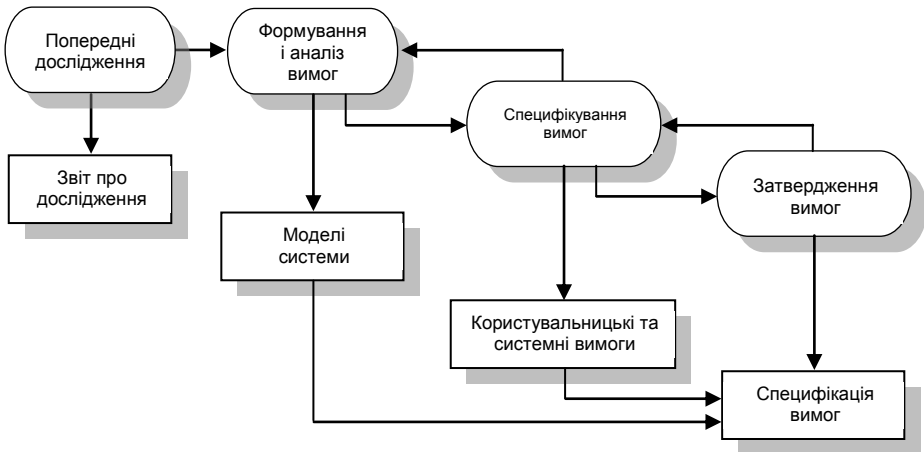


Рисунок 1.3 - Процес розробки вимог

Процес розробки вимог включає чотири основні етапи.

1. Попередні дослідження. Оцінюється ступінь задоволеності користувачів існуючими програмними продуктами і апаратними засобами, а також економічна ефективність майбутньої системи і можливість укластися в існуючі бюджетні обмеження при її розробці. Цей етап повинен бути по можливості коротким і дешевим.

2. Формування і аналіз вимог. Формуються системні вимоги шляхом вивчення існуючих аналогічних систем, обговорення майбутньої системи з потенційними користувачами і замовниками, аналізу задач, які повинна вирішувати система, і т.і. Цей етап може включати розробку декількох моделей системи і її прототипів, що допомагає сформувати функціональні вимоги до системи.

3. Специфікування вимог. Здійснюється переклад всієї сукупності інформації, зібраної на попередньому етапі, в документ, що визначає множину вимог. Цей документ звичайно містить два типи вимог: призначені для користувача - узагальнені уявлення замовників і кінцевих користувачів про систему; системні - детальний опис функціональних показників системи.

4. Затвердження вимог. Перевіряється здійснимість, узгодженість і повнота множини вимог. В процесі формування обмежень неминуче виникнення яких-небудь помилок. На цьому етапі вони повинні бути по можливості виявлені і усунені.

Звичайно, процес розробки вимог важко укласти в описану послідовність етапів. Наприклад, аналіз вимог виконується впродовж всього процесу їх розробки, тому внесення нових або зміна вже сформульованих вимог можливе на будь-якому етапі. Як правило, етапи розробки вимог перекриваються в часі.

Деякі проблеми, що виникають в процесі розробки вимог, породжені відсутністю чіткого розуміння відмінності між цими різними рівнями вимог. Щоб розрізнити вимоги різних рівнів використовуються терміни користувальницькі вимоги (user requirements) для позначення високорівневих узагальнених вимог і системні вимоги (system requirements) для опису виконуваних системою функцій, що деталізуються. Окрім вимог цих двох рівнів, застосовується ще опис системи, що більш деталізується - проектна системна специфікація (software design specification), яка може служити мостом між етапом розробки вимог і етапом проектування системи. Три перераховані види вимог можна визначити таким чином.

1. Користувальницькі вимоги - опис на природній мові (плюс пояснюючі діаграми) функцій, виконуваних системою, і обмежень, що накладаються на неї.

2. Системні вимоги - опис системних функцій і обмежень, який іноді називають функціональною специфікацією, що деталізується. Вона служить основою для укладення контракту між покупцем системи і розробниками ПЗ.

3. Проектна системна специфікація - узагальнений опис структури програмної системи, який буде основою для проектування системи, що більш деталізується, і її подальшої реалізації. Ця специфікація доповнює і деталізує специфікацію системних вимог.

Призначені для користувача вимоги пишуться для замовника ПЗ і для особи, що укладає контракт на розробку програмної системи, причому вони можуть не мати детальних технічних знань по системі, що розробляється (рис. 1.4). Специфікація системних вимог призначена для керівного технічного складу компанії-розробника і для менеджерів проекту. Вона також необхідна замовнику ПО і субпідрядникам по розробці. Ці обидва документи також призначено

для кінцевих користувачів програмної системи. Нарешті, проектна системна специфікація є документом, який орієнтований на розробників ПО.

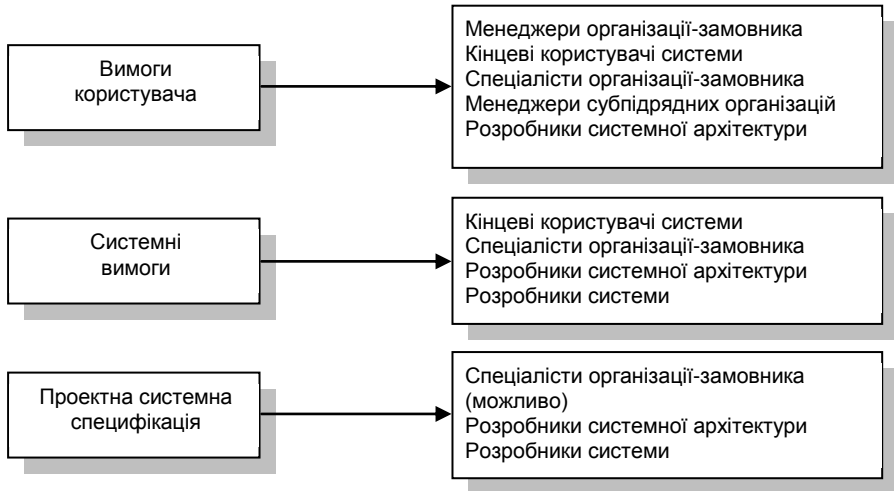


Рисунок 1.4 - Різні типи специфікацій вимог і їх читачі

Вимоги до програмної системи часто класифікуються як функціональні, нефункціональні і вимоги предметної області.

1. Функціональні вимоги. Це перелік сервісів, які повинна виконувати система, причому повинно бути вказано, як система реагує на ті або інші вхідні дані, як вона поводить себе в певних ситуаціях і т.д. В деяких випадках указується, що система не повинна робити.

2. Нефункціональні вимоги. Описують характеристики системи і її оточення, а не поведінка системи. Тут також може бути приведений перелік обмежень, що накладаються на дії і функції, виконувані системою. Вони включають часові обмеження, обмеження на процес розробки системи, стандарти і т.д.

3. Вимоги предметної області. Характеризують ту предметну область, де експлуатуватиметься система. Ці вимоги можуть бути функціональними і нефункціональними.

Насправді чіткої межі між цими типами вимог не існує. Наприклад, призначені для користувача вимоги, що стосуються

безпеки системи, можна віднести до нефункціональних. Проте при більш детальному розгляді таку вимогу можна віднести до функціональних, оскільки воно породжує необхідність включення в систему засобу авторизації користувача. Тому, розглядаючи далі ці види вимог, ми повинні завжди пам'ятати, що дана класифікація в значній мірі штучна.

1.5.2 Запитання для самоперевірки

- 1.5.2.1 В чому призначення специфікації?
- 1.5.2.2 Які основні етапи процесу розробки специфікації?
- 1.5.2.3 Що визначають вимоги користувача?
- 1.5.2.4 Що визначають системні вимоги?
- 1.5.2.5 Що являє собою проектна системна специфікація?
- 1.5.2.6 Для кого призначені вимоги користувача?
- 1.5.2.7 Для кого призначені системні вимоги?
- 1.5.2.8 Для кого призначена проектна системна специфікація?

1.6 Документування ПЗ. Види програмних документів

Розробка документації на ПЗ. Види програмних документів. Технічне завдання. Пояснювальна записка. Програма і методика випробувань. Опис програми. Опис застосування. Опис мови. Керівництво програміста. Керівництво оператора. [2(с. 23-29)].

1.6.1 Методичні вказівки

Розробка документації на ПЗ – відповідальний етап проектування. Від якості виконання документації залежить не тільки ефективність використання програмних засобів, але і придатність їх до розвитку і супроводу. Документація повинна створюватися і коректуватися в процесі проектування. Документування ПЗ здійснюється відповідно до Єдиної Системи Програмної Документації (ЄСПД). ЄСПД – являє собою комплекс державних стандартів, що встановлюють правила розробки, оформлення і обігу програм і програмної документації. В стандартах ЄСПД встановлені вимоги, що регламентують розробку, супровід, виготовлення та експлуатацію програм, що дає можливість уніфікації програмних виробів для взаємного обміну програмами, зниження трудомісткості і підвищення ефективності розробки, супроводу, виготовлення та експлуатації

програмних виробів; автоматизації виготовлення та збереження програмної продукції.

ЄСПД встановлює наступні види програмних документів:

- специфікацію, що містить склад програми та документацію на неї;
- відомість власників оригіналів – перелік підприємств, на яких зберігають оригінали програмних документів;
- технічне завдання, що визначає вимоги до програми, необхідні стадії та терміни розробки, види випробувань;
- пояснювальну записку, що містить загальний опис алгоритму й функціонування програми;
- програму та методику випробувань, що містять вимоги, що підлягають перевірці при випробуванні програми, а також порядок та методи їхнього контролю;
- опис програми – відомості про логічну структуру і функціонування програми;
- текст програми – запис програми з необхідними коментарями;
- експлуатаційні документи, що містять:
 - відомість експлуатаційних документів – перелік експлуатаційних документів на програму;
 - формуляр – визначає основні характеристики програми, комплектність і відомості про експлуатацію;
 - опис застосування – містить відомості про призначення, область застосування програми, методи та класі розв'язуваних задач;
 - керівництво програміста – містить відомості, що необхідні для експлуатації програми;
 - керівництво системного програміста – містить відомості для перевірки, забезпечення функціонування та налаштування програми на рівні конкретного додатка;
 - опис мови – визначає синтаксис і семантику мови взаємодії користувача з програмою.
 - керівництво оператора – містить відомості, необхідні для забезпечення процедури спілкування оператора з ЕОМ у процесі виконання програми;
 - посібник із технічного обслуговування – містить опис застосування тестових і діагностичних програм при обслуговуванні технічних засобів.

1.6.2 Запитання для самоперевірки

- 1.6.2.1 Для чого потрібне документування ПЗ?
- 1.6.2.2 Що таке ЄСПД і в чому її призначення?
- 1.6.2.3 Перелічите види програмних документів.
- 1.6.2.4 Перелічите види експлуатаційних документів.
- 1.6.2.5 Які розділи включає технічне завдання?
- 1.6.2.6 Які розділи включає пояснювальна записка?
- 1.6.2.7 Які розділи включає програма і методика іспитів?
- 1.6.2.8 Які розділи включає опис програми?
- 1.6.2.9 Які розділи включає опис застосування?
- 1.6.2.10 Які розділи включає керівництво програміста?
- 1.6.2.11 Які розділи включає опис мови?
- 1.6.2.12 Які розділи включає керівництво оператора?

1.7 Проектування модульної структури ПЗ. Основні структури модульних програм

Принцип модульного програмування. Вихідний модуль. Об'єктний модуль. Завантажувальний модуль. Монолітно-модульна структура. Модульно-послідовна структура. Модульно-ієрархічна структура. Модульно-хаотична структура. [2(с. 42-46)].

1.7.1 Методичні вказівки

Принцип модульного програмування полягає в поділі програм на функціонально самостійні частини (модулі), що забезпечують заміність, модифікацію, видалення і доповнення складових частин (модулів). Переваги використання модульного принципу програмування полягають у наступному:

- Спрощується налагодження програм, тому що обмежений доступ до модуля й однозначність його зовнішнього поводження виключають вплив помилок в інших, зв'язаних з ним, модулях на його функціонування. Це дає можливість розробляти програму методами зверху вниз чи знизу нагору, поступово приєднуючи написані модулі до раніше налагодженого. Після кожного такого приєднання невірна робота програми сигналізує про присутність помилки в новому модулі, а не вже налагоджених.

- Забезпечується можливість організації спільної роботи великих колективів розроблювачів, тому що кожен програміст має справу з незалежною від інших частиною програми (модулем або групою модулів).

- Підвищується надійність програми, тому що відносно малий розмір модулів і, як наслідок, їхня невелика складність дозволяють провести більш повну перевірку програми.

У системах програмування розрізняються три основних види модулів, що відповідають основним етапам одержання готової програми: вихідний, об'єктний, завантажувальний. Вихідний модуль містить команди обраної мови програмування. Надалі він підлягає компіляції. У результаті компіляції виходить об'єктний модуль. Об'єктний модуль складається з тексту машинною мовою і містить, крім машинних команд і констант, керуючі словники, що необхідні для наступного завантаження та настроювання програми в оперативній пам'яті. Об'єктні модулі мають стандартний формат, незалежно від вихідної мови програмування. Спеціальною програмою – редактором – вони поєднуються в завантажувальний модуль. Завантажувальний модуль – програма, що складається з тексту машинною мовою, і керуючих словників, що необхідні для його завантаження в оперативну пам'ять для наступного виконання. Програміст має справу в основному з вихідними модулями.

З метою дотримання принципу розвитку ПЗ проектування модульної структури програм будують по модульному принципу. Функціональна самостійність модуля досягається за рахунок раціональної розбивки функцій обробки даних на підфункції та оформлення їх у вигляді окремо компілюємих частин. Модульна структура програми розробляється на стадії технічного проекту. Результатам проектування модульної структури є визначення складу програмних модулів і встановлення зв'язків між ними по керуванню та даним.

У залежності від задач, розв'язуваних розроблювачами, та від використання ними методів проектування модульна програма може мати одну з наступних основних структур: монолітно-модульну, модульно-послідовну, модульно-ієрархічну, модульно-хаотичну.

Монолітно-модульна структура (рисунк 1.5, а) містить у собі великий програмний модуль, що реалізує основну частину покладених на програму функцій. З цієї частини мається незначне число звертань

до інших програмних модулів невеликого розміру. Подібна програма несе в собі всі основні недоліки не модульної, монолітної програми: вона складна для розуміння, перевірки, супроводу. Одержання подібної структури програми варто уникати при проектуванні САПР. Усі програми модулі рекомендується обмежувати за розміром 100 операторами вихідної мови програмування.

Модульна-послідовна структура (рисунок 1.5, б) містить у собі декілька програмних модулів, що послідовно передають керування один одному. Структура досить проста і наочна, але може бути реалізована тільки для відносно простих задач.

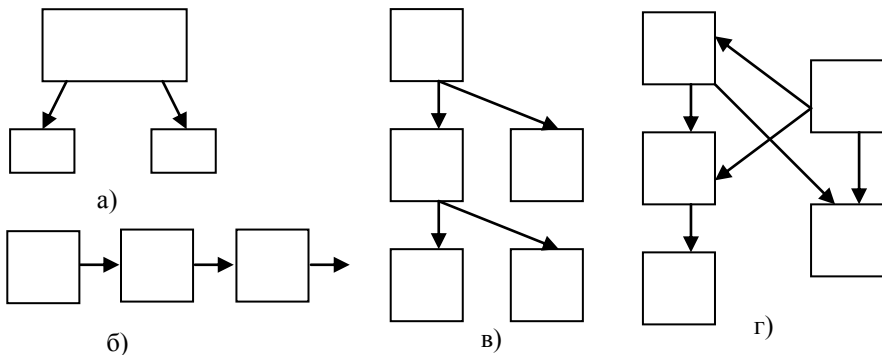


Рисунок 1.5 – Основні види модульних структур

Модульна-ієрархічна структура (рисунок 1.5, в) містить у собі програмні модулі, розташовані на декількох рівнях ієрархії. Модулі верхніх рівнів керують роботою модулів нижніх рівнів. Вищестоящий модуль передає керування модулю більш низького рівня, а коли той відробить, він повертає керування модулю, що його викликав. Подібна структура досить проста і дозволяє вирішувати дуже складні задачі.

Модульна-хаотична структура (рисунок 1.5, г) містить у собі програмні модулі, що зв'язані між собою таким чином, що вони не утворюють у явному вигляді ні одну з перерахованих вище структур. Такі програми складні для перевірки та супроводу. Слід по можливості уникати одержання модульно-хаотичних програм. Вони можуть виявитися припустимими тільки в системах реального часу з жорсткими об'ємно-часовими обмеженнями, коли за допомогою

програм інших видів неможливо буває досягти заданих характеристик.

1.7.2 Запитання для самоперевірки

1.7.2.1 В чому полягає принцип модульного програмування?

1.7.2.2 В чому є переваги використання модульного принципу програмування?

1.7.2.3 Що таке вихідний модуль?

1.7.2.4 Що таке об'єктний модуль?

1.7.2.5 Що таке завантажувальний модуль?

1.7.2.6 Назвіть основні структури модульних програм?

1.7.2.7 Що являє собою монолітно-модульна структура?

1.7.2.8 Що являє собою модульно-послідовна структура?

1.7.2.9 Що являє собою модульно-ієрархічна структура?

1.7.2.10 Що являє собою модульно-хаотична структура?

1.8 Структурування програмних компонентів. Висхідне та спадне проектування ПЗ. Елементарні базові структури

Структура комплексу програм. Оператори мови програмування. Програмні модулі. Функціональні групи програм. Комплекси програм. Проектування зверху вниз. Проектування знизу нагору. Елементарні базові структури. Заборонені структури [1(с. 47-49, 53-54)].

1.8.1 Методичні вказівки

При ієрархічній побудові КП важливе значення мають обсяг, складність компонентів для кожного рівня ієрархії і, відповідно, кількість ієрархічних рівнів для визначених класів КП. За принципом побудови, мові опису, обсягу й іншим характеристикам у структурі КП можна виділити наступні ієрархічні рівні: операторів і операндів програми, що відповідають компонентам тексту програми мовою програмування; програмних модулів, оформлюваних як закінчені компоненти тексту програми; функціональних груп програм або пакетів прикладних програм; комплексу програм, оформлюваного як завершений ПЗ визначеного цільового призначення. З підвищенням ієрархічного рівня збільшується кількість машинних команд комп'ютера, що реалізують компоненти кожного рівня та кількість оброблюваних змінних. Одночасно сукупності команд усе більш

спеціалізуються, тобто знижується можливість застосування компонент у різних комбінаціях для рішення аналогічних задач.

Оператори мови програмування є мінімальними компонентами, із яких будуються модулі. Розмаїтість операторів порівняно невелика (50-100 типів), і кожен оператор реалізується алгоритмом на базі в середньому 1-10 машинних команд комп'ютера. З підвищенням рівня мови програмування зростає функціональна складність операторів. Програмні модулі вирішують невелику функціональну задачу і реалізуються 10-100 операторами мови програмування високого рівня або 100-1000 операторами асемблера. У результаті програма модуля має 100-1000 машинних команд. Кожен модуль може використовувати на вході біля десятка типів змінних, але зустрічаються програмні модулі, що обробляють кілька десятків типів операндів. Кількість типів вихідних даних трохи менше. Якщо для рішення невеликої функціональної задачі потрібно більш 100 операторів, то доцільно провести декомпозицію задачі на декілька більш простих, для реалізації, кожної з яких модуль реалізується 50-100 операторами. Функціональні групи програм і пакети прикладних програм формуються на базі десятків модулів і вирішують складні автономні функціональні задачі. На їхню реалізацію використовується близько 10000 команд. Відповідно зростає кількість використаних типів змінних і розмаїтість вихідних даних. При цьому значно швидше зростає кількість типів змінних, що обробляються модулями і локалізуються в межах одного чи декількох модулів. Комплекси програм створюються для рішення особливо складних задач керування й обробки інформації або обчислювальних задач у науці та техніці. В комплекси поєднуються декілька або десятки груп програм для рішення загальної цільової задачі. Розміри КП обчислюються сотнями модулів, десятками й сотнями тисяч машинних команд. Зустрічаються КП, що містять до двох-трьох десятків структурних ієрархічних рівнів, побудованих з модулів.

Ієрархічна багаторівнева побудова КП значно полегшує організацію їхнього проектування та експлуатації, скорочує тривалість і вартість розробки. З'являється можливість раціонально розподіляти зусилля розроблювачів на рішення приватних задач відповідно до їхньої важливості для основного цільового призначення програм з урахуванням кваліфікації кожного фахівця. По кількості компонентів на різних рівнях з урахуванням складності зв'язків можна

оцінювати обсяг виконуваної роботи та прогнозувати його перспективи по термінах і трудомісткості, у наслідок чого підвищується вірогідність контролю стану та процесу проектування.

Багаторівневий ієрархічний підхід дозволяє проектувати складні КП за принципом зверху-вниз з позиції призначення та отримання рішення основної цільової задачі всієї системи. Це забезпечує концептуальну єдність програм і можливість раціонального розподілу ресурсів проектування в міру декомпозиції компонент. Хоча поділ на ієрархічні рівні вимагає деяких витрат, у цілому ресурси використовуються більш ефективно, ніж при відсутності чіткої ієрархії за рахунок ощадливої побудови та спрощення компонент на кожному рівні.

Іноді основне проектування зверху-вниз супроводжує розробка компонент знизу-нагору. Розробка починається з модулів наступного рівня ієрархії і т.д. Перевагою цього принципу є те, що при переході до розробки модулів більш високого рівня ієрархії модулі нижніх рівнів можна вважати готовими і підключати їх до модулів верхнього рівня на стадії налагодження. Однак на практиці при такому підході відсутність цілісного погляду на КП із позицій верхнього рівня, що визначає мету побудови комплексу, не дозволяє в ряді випадків приймати вірні рішення, що приводить до повторної розробки або значного коректування модулів. Вплив повторної розробки позначається тим важче, чим вище рівень ієрархії, на якому виявлена помилка. Розробка КП цілком за принципом знизу-нагору можлива лише для порівняно невеликих груп програм, обмежених декількома модулями, коли розроблювачі здатні оцінювати в будь-який час структуру КП в цілому та структуру й функції окремих модулів на всіх рівнях ієрархії. Тому при розробці складних КП, що містять сотні модулів, найбільш раціональним принципом є проектування зверху-вниз.

Існує ряд програмних конструкцій, що при перекручуванні вихідних даних можуть приводити до катастрофічних наслідків. Найбільш відомою, важко контрольованою та потенційно ненадійною конструкцією є безумовний перехід за вмістом осередку оперативної пам'яті (оператор GOTO). Тому структура тіла модулів та використовувані базові конструкції повинні бути потенційно стійкими до апаратних збоїв, перекручувань вихідних даних та помилок у програмах.

Будь-яку програму можна синтезувати на основі елементарних базових конструкцій трьох типів: простої обчислювальної послідовності, вибору або альтернативи та повторення або ітерації. Кожна структура характеризується єдиною точкою передачі керування в структуру та єдиною точкою виходу зі структури. Проста обчислювальна послідовність полягає в перетворенні або переміщенні сукупності вихідних змінних. Елементарні конструкції ідуть друг за другом, причому кінець попереднього оператора замикається на початок наступного (рисунок 1.6, а).

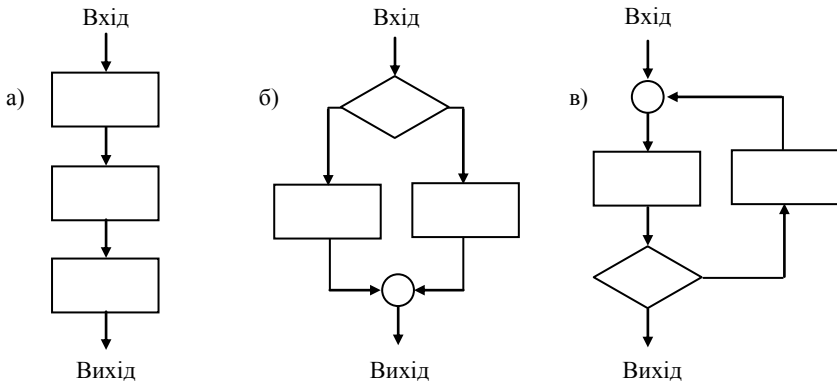


Рисунок 1.6 – Елементарні базові структури

Альтернатива складається в перевірці виконання деякої умови та вибору одного з двох операторів перетворення даних в залежності від реалізації умови. При розгалужуванні здійснюється однократний прохід по одній з галузей рішення задачі (рисунок 1.6, б). Ітерація являє собою структуру, в якій при кожному обертанні один чи кілька операторів повторюються більш одного разу. Для структурованих програм число ітерацій повинно бути вказано на вході в цикл, а не визначатися обчисленнями усередині циклу. В результаті виключається можлива невизначеність функціонування програми (рисунок 1.6, в).

Ці конструкції мають систематизуюче та дисциплінуюче значення. Простота вихідних конструкцій структурного програмування запобігає появі складних інформаційних зв'язків та заплутаних передач керування. Структурованими вважаються

програми, що не мають циклів з декількома виходами, не мають переходів усередину циклів або умовних операторів та не мають виходів із внутрішньої частини циклів або умовних операторів. Таким забороненим структурам відповідають компоненти програм, представлені на рисунку 1.7, а-г. При підвищенні структурованості модулів знижується складність програм, зростає їхня наочність, що сприяє скороченню числа помилок. Однак за підвищення якості програм приходится розплачуватися додатковою пам'яттю та часом їхньої реалізації на комп'ютері.

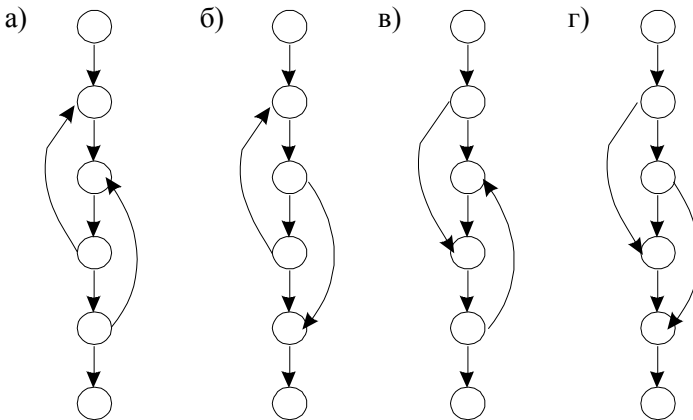


Рисунок 1.7 – Заборонені структури

1.8.2 Запитання для самоперевірки

1.8.2.1 Які ієрархічні рівні можна виділити в структурі КП?

1.8.2.2 Чим з точки зору структури КП є оператори мови програмування?

1.8.2.3 Яке місце в ієрархії структури КП займають програмні модулі та яке їхнє призначення?

1.8.2.4 Яке місце в ієрархії структури КП займають функціональні групи програм та яке їхнє призначення?

1.8.2.5 В чому призначення та особливості комплексів програм з точки зору структури програмних компонентів?

1.8.2.6 В чому особливості проектування КП за принципом зверху-вниз?

1.8.2.7 В чому особливості проектування КП за принципом знизу-нагору?

1.8.2.8 Що таке елементарні базові структури та в чому їхнє призначення?

1.8.2.9 Що таке заборонені структури?

1.9 Надійність функціонування комплексів програм. Фактори, що знижують надійність функціонування програм. Програмні методи підвищення надійності функціонування КП

Відмовлення. Збій. Перекручування вихідної інформації. Збої в апаратурі КС. Помилки в програмах. Захист від зациклень. Захист від зупинки. Захист від перекручувань взаємного переривання програм. Захист від помилок, що приводять до пропуску програм. Захист від перевантаження КС за пропускну здатністю. Захист від перекручування та втрати накопиченої інформації про стан зовнішнього середовища [1(с. 173-185), 3(16-24)].

1.9.1 Методичні вказівки

Надійність технічних систем визначається в основному двома факторами: надійністю компонент та помилками в конструкції, допущеними при проектуванні і виготовленні. Надійність складних КП визначається тими ж двома факторами, однак на відміну від апаратури, де головним фактором є надійність компонентів (через старіння, зниження надійності в процесі експлуатації) для надійності КП головним фактором є помилки проектування.

Джерелами ненадійності є неперевірені сполучення вихідних даних, при яких налагоджений КП дає невірні результати або відмови. Поняття "відмова" пов'язано з порушенням працездатності виробу та його відповідності вимогам технічної документації. Відмова при виконанні програм може з'явитися як наслідок:

- стирання або перекручування даних в оперативній чи довгостроковій пам'яті комп'ютера;
- порушення кодів запису команд програм у пам'яті;
- порушення нормального ходу обчислювального процесу.

В усіх випадках програмні відмови приводять до припинення видачі користувачам інформації та керуючих впливів або до значного перекручування її змісту й темпів видачі.

Поняття "збій" у теорії надійності трактується як відмовлення, що самоусувається та не вимагає зовнішнього втручання. Основний принцип класифікації збоїв та відмов - розподіл по часовому показнику тривалості відновлення після будь-якого переключування програми, або даних обчислювального процесу. При тривалості відновлення, меншої заданого порогу при роботі програм варто відносити до збоїв. При відновленні, що перевищує по тривалості граничне значення та порушує працездатність програм, переключування відповідають відмові.

Головною задачею стає відновлення за час, що не перевищує граничне значення між збоєм та відмовою. В результаті можна перетворити відмови на збої і тим самим поліпшити показники надійності функціонування системи. Для рішення цієї задачі в КП повинні бути засоби, що дозволяють:

- проводити систематичний контроль та виявляти аномалії процесу функціонування або стану програм і даних;
- діагностувати отримані переключування;
- вибирати методи та засоби оперативного відновлення (рестарту);
- реалізовувати оперативне відновлення нормальної працездатності;
- ресетрувати кожен збій, чи відмову та узагальнювати з даними попередніх переключувань для виявлення систематичних випадків, що вимагають доробки програм або апаратури.

Реалізація засобів з такими функціями за рахунок введення надмірності в програми, дані та процес функціонування КП:

- програмної, що включає всі програмні компоненти, що призначені для контролю, виявлення, діагностики та відновлення працездатності КП;
- інформаційної, що полягає в дубльованому збереженні даних та засобах кодового захисту інформації;
- часової, що складається у виділенні необхідних резервів процесорного часу КС на виконання програм, що забезпечують оперативний контроль і відновлення (рестарт) функціонування КП.

На надійність функціонування КП впливають насамперед фактори, що викликають збій або відмову при виконанні програми:

- переключування вихідної інформації, що надходить від зовнішніх абонентів;

- відмовлення, що самоусуваються, чи збої в апаратурі КС;
- не виявлені помилки в програмах.

Перекручені дані в ряді випадків можуть бути причиною тільки помилок у результатах, що видаються зовнішнім абонентам або накопичуються в пам'яті КС, і не впливають на надійність КП. Однак деякі перекручування виходять за область припустимих значень змінних. При цьому зростає ймовірність того, що перекручена величина буде оброблятися деяким сполученням команд, що приводить або до відмови, або до збою функціонування.

Захист від зациклень у програмах запобігає перекручуванню реальних підготовлених циклів, а також утворенню непередбачених, помилкових циклів. Характерною ознакою зациклення є перевищення тривалості розрахунку за програмою максимально припустимого часу її реалізації при нормальній роботі. Для виключення помилкових явищ зациклення, цей гранично припустимий час повинний враховувати самий тривалий маршрут реалізації програми при наявності переривань іншими програмами, що виконуються з абсолютним пріоритетом.

Автоматичне виявлення зациклення найбільш просто робити при наявності в складі апаратури КС лічильника відносного часу для підрахунку тривалості часових інтервалів. У цьому випадку перед включенням групи програм на лічильнику відносного часу встановлюється гранично припустимий контрольний час їхньої реалізації. У процесі роботи КС показання лічильника рівномірно зменшуються, що забезпечується рівномірною позапрограмною подачею періодичних сигналів точного часу. При досягненні нульового значення на лічильнику виробляється сигнал перевищення припустимого контрольного часу (наявність зациклення). Цей сигнал забезпечує переривання програми та передачу керування програмі для аналізу причин аномально тривалого розрахунку та ухвалення рішення про відновлення нормальних обчислень. Якщо при реалізації групи програм не порушене обмеження за часом, то лічильник не досягає нульового значення і при передачі керування черговій програмі на ньому встановлюється відповідна нова гранична тривалість її реалізації.

Причиною зациклення можуть бути не тільки помилки в програмі та перекручування вихідної інформації, але і збої в апаратурі КС. Вони характеризуються малою ймовірністю повторення при

наступній реалізації програми при тих же вихідних даних. Тому при виявленні первинного зациклення варто повторити включення програми при тих же вихідних даних, якщо вони не були зіпсовані під час зациклення. Причиною багаторазових зациклень із різними вихідними даними є швидше за все часткове відмовлення в апаратурі або перекручування інформації в процес керування. Тоді варто перевірити функціонування на резервній машині.

Захист від зупинки за методами принципово близький до захисту від зациклення. Зупинка КС відбувається або через помилку при форматуванні команди в КС (часткова відмова або збій), або через помилку в програмі, що приводить до потрапляння на ділянку програми, що містить команду зупинки. Автоматичне виявлення зупинки може робитися аналогічно виявленню зациклення.

Захист від перекручувань взаємного переривання програм, що приводять до можливості блокування деяких типів переривань здійснюється в основному апаратними методами. Для захисту від таких програмних помилок, а так само від апаратних збоїв при перериваннях повинні передбачатися програмний контроль виконання переривань та періодичний контроль наявності взаємодії з усіма абонентами. Для виконання цих функцій спеціальна періодична програма повинна підготувати контрольні повідомлення та видавати їх абонентам, а також аналізувати періодичний прийом від абонентів контрольних повідомлень.

Захист від помилок, що приводять до пропуску програм чи їхніх істотних частин, виробляється в основному методами контролю. При контролі за допомогою ключових кодів записується таблиця імен програм, що мали бути обов'язково включені, та кількість включень кожної. Після реалізації кожної програми з таблиці вилучається її код, і таким чином можна установити чи всі необхідні програми включалися відповідну кількість разів.

При виявленні пропуску програм при її завершенні варто зробити повторне включення усієї функціональної групи. В окремих випадках може бути корисним автоматичне примусове включення пропущеної програми.

Захист від перевантаження КС за пропускнуою здатністю припускає виявлення та зниження впливу алгоритмічних помилок, обумовлених неправильним визначенням необхідної пропускнуої здатності КС для роботи в реальному часі. Крім того, перевантаження

можуть бути наслідком неправильного функціонування джерел інформації та перевищення інтенсивності потоків повідомлень розрахункового, нормального рівня. Наслідки звичайно, зводяться до припинення рішення деяких функціональних задач, що володіють низьким пріоритетом.

Для захисту від перевантаження необхідно привести до відповідності потребу в продуктивності КС для рішення задач із реальною продуктивністю КС. Потребу в продуктивності КС можна знизити шляхом диференційованого проріджування потоків повідомлень і збільшенням періоду включення деяких періодичних програм. Реальну продуктивність КС можна збільшити підключенням дублюючих комп'ютерів. Програмні методи захисту можуть істотно знизити вплив перевантажень шляхом деякого перерозподілу ресурсів КС та адаптацією програм на рівень реального завантаження КС.

Захист від перекручувань і втрати накопиченої інформації про стан зовнішнього середовища передбачає контроль результатів перед їхнім переписом і в процесі перепису в зони довгострокового збереження інформації, а також захист цих зон від випадкового запису в них інформації програмами, не призначеними для виконання цієї операції. Групи програм доцільно будувати з двох частин: “функціональної” та “посилочної”. “Посилочна” частина робить контроль результатів і перевіряє відповідність “паролів” та адрес зон довгострокового збереження накопиченої інформації, куди потім переписуються результати, отримані функціональною частиною.

Контроль втрати та перекручувань накопиченої інформації про стан зовнішнього середовища робиться в процесі її використання при рішенні функціональних задач перед її відновленням та періодично шляхом перевірки контрольних сум по збереженим змінним. При виявленні перекручування інформації можливі наступні рішення: продовження обчислень і поступове відновлення перекрученої інформації; відновлення загубленої інформації шляхом використання дублюючих даних із резервної КС; перехід на резервну КС, що містить накопичену інформацію.

1.9.2 Запитання для самоперевірки

1.9.2.1 Який фактор є головним для надійності КП?

1.9.2.2 Що розуміється під поняттям “відмова”?

1.9.2.3 Що є причинами відмов при виконанні програм?

1.9.2.4 Що розуміється під поняттям “збій”?

1.9.2.5 Які види надмірності вводяться в програми, дані і процес функціонування КП для поліпшення показників надійності?

1.9.2.6 Перелічите фактори, що знижують надійність функціонування програм.

1.9.2.7 Перелічите програмні методи підвищення надійності функціонування КП.

1.9.2.8 В чому полягає захист від зациклень?

1.9.2.9 В чому полягає захист від зупинки?

1.9.2.10 В чому полягає захист від перекручувань взаємного переривання програм?

1.9.2.11 В чому полягає захист від помилок, що приводять до пропуску програм або їхніх істотних частин?

1.9.2.12 В чому полягає захист від перевантаження КС за пропусковою здатністю?

1.9.2.13 В чому полягає захист від перекручування та втрати накопиченої інформації про стан зовнішнього середовища?

1.10 Супровід програмного забезпечення

Види супроводу ПЗ. Спіральна модель розвитку ПЗ. Фактори, що впливають на вартість супроводу ПЗ. Шляхи зменшення вартості супроводу ПЗ. [4(с. 554-562)].

1.10.1 Методичні вказівки

Супровід - це звичайний процес зміни системи після її поставки замовнику. Ці зміни можуть бути як елементарно простими (виправлення помилок програмування), так і більш серйозними, пов'язаними з коригуванням окремих недоробок або приведенням у відповідність з новими вимогами. Супровід не пов'язаний зі значною зміною архітектури системи. При супроводі тактика проста: зміна існуючих компонентів системи або додавання нових.

Існує три види супроводу системи.

1. Супровід з метою виправлення помилок. Зазвичай помилки в програмуванні досить легко переборні, однак помилки проектування коштують дорого і вимагають коректування або перепрограмування деяких компонентів. Найдорожчі виправлення пов'язані з помилками в

системних вимогах, так як тут може знадобитися перепроєктування системи.

2. Супровід з метою адаптації ПЗ до специфічних умов експлуатації. Це може знадобитися при зміні певних складових робочого оточення системи, наприклад апаратних засобів, операційної системи або програмних засобів підтримки. Щоб адаптуватися до цих змін, система повинна бути піддана певним модифікаціям.

3. Супровід з метою зміни функціональних можливостей системи. У відповідь на організаційні або ділові зміни в організації можуть змінитися вимоги до програмних засобів. У таких випадках застосовується даний тип супроводу. Найбільш істотні зміни при цьому зазнає саме програмне забезпечення.

На практиці однозначно чітке розмежування між різними видами супроводу провести досить складно. Помилки в системі можуть бути виявлені в тому випадку, якщо, наприклад, система використовувалася непередбачуваним способом. Тому найкращий спосіб виправлення помилок - розширення функціональних можливостей програми з тим, щоб зробити роботу з нею якомога простіше. При адаптації програмного забезпечення до нового робочого оточення розширення функціональних можливостей системи сприятиме поліпшенню її роботи. Також додавання певних функцій в програму може виявитися корисним, якщо в разі помилок був змінений шаблон використання системи і побічною дією при розширенні функціональних можливостей буде видалення помилок.

Згідно з дослідженнями 65% супроводу пов'язано з виконанням нових вимог 18% відводиться на зміни системи з метою адаптації до нового оточення і 17% пов'язано з виправленням помилок (рис. 1.8).



Рисунок 1.8- Розподіл типів супроводу

З цього можна визначити, що виправлення помилок не є найпоширенішим видом супроводу. Модернізація системи відповідно до нового робочим оточенням або відповідно до нових вимог більш ефективна. Тому супровід само по собі є природним процесом продовження розробки, системи зі своїми процесами проектування, реалізації та тестування. Таким чином, спіральна модель, показана на рис. 1.9, краще представляє процес розвитку ПЗ, ніж каскадна модель (див. рис. 1.2.), де супровід розглядається як окремий процес.

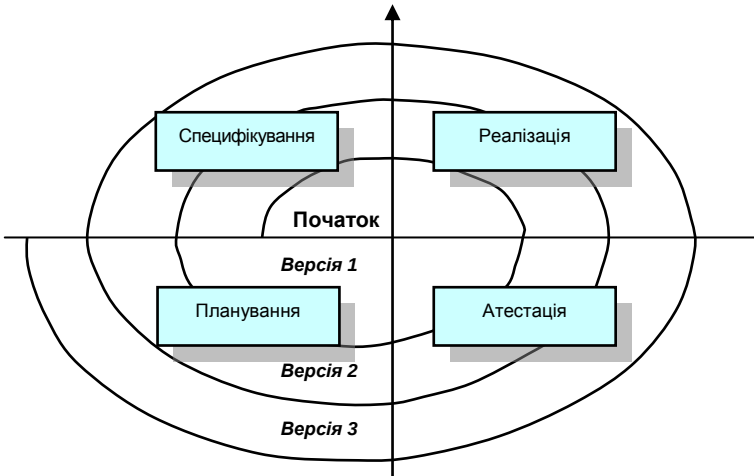


Рисунок 1.9 - Спіральна модель розвитку ПЗ

Співвідношення між величинами коштів на супровід і на розробку можуть бути різними в залежності від предметної області, де експлуатується система. Для прикладних систем, що працюють в діловій сфері, співвідношення витрат на супровід в основному порівняно із засобами, витраченими на розробку. Для вбудованих систем реального часу витрати на супровід можуть в чотири рази перевищувати вартість самої розробки. Високі вимоги щодо продуктивності і надійності таких систем припускають їх жорстку структуру, яка важче піддається модифікації.

Причиною високих витрат на супровід є складність модернізації системи після її впровадження, оскільки розширити функціональні можливості набагато легше в процесі створення системи. Нижче

наведені ключові фактори, які визначають вартість розробки і супроводу і можуть привести до подорожчання супроводу.

1. Стабільність команди розробників. Цілком природно, що після впровадження системи команда розробників розпадається, фахівці працюватимуть над іншими проектами. Новим членам команди або ж окремим фахівцям, які візьмуть на себе подальший супровід системи, буде важко зрозуміти всі її особливості. Тому на розуміння системи перед внесенням до неї змін йде багато часу і коштів.

2. Відповідальність згідно з контрактом. Контракт на супровід зазвичай укладається окремо від договору на розробку програми. Більш того, часто контракт на супровід може отримати фірма, сама не займається розробкою. Разом з фактором нестабільності команди це може стати причиною відсутності в команді стимулу створити систему, що легко змінюється та зручна в супроводі. Якщо членам команди вигідно піти найкоротшим шляхом з мінімальними витратами зусиль, то навряд чи вони відмовляться від цього навіть з ризиком підвищення наступних витрат на супровід.

3. Кваліфікація фахівців. Фахівці, що займаються супроводом, часто не знайомі з предметною областю, де експлуатується система. Супровід не користується популярністю серед розробників. Це вважається менш кваліфікованою розробкою і часто доручається молодшому персоналу. Більш того, старі системи можуть бути написані на застарілих мовах програмування, необізнаних молодим фахівцям і вимагають додаткового вивчення.

4. Вік і структура програми. З віком структура програм порушується внаслідок частих змін, тому їх стає складніше розуміти і змінювати. Крім того, багато успадковані системи були створені без використання сучасних технологій. Вони ніколи не відрізнялися хорошою якісною структурою; зміни, зроблені в них, були спрямовані скоріше на підвищення ефективності функціонування, ніж на підвищення зручності супроводу. Документація на старі системи часто буває неповною або взагалі відсутня.

1.10.2 Запитання для самоперевірки

1.10.2.1 Які існують види супроводу ПЗ та яке їх процентне співвідношення?

1.10.2.2 В чому є суть супроводу з метою виправлення помилок?

1.10.2.3 В чому є суть супроводу з метою адаптації ПЗ?

1.10.2.4 В чому є суть супроводу з метою розширення функціональних можливостей ПЗ?

1.10.2.5 Що таке спіральна модель розвитку ПЗ?

1.10.2.6 Чому вартість супроводу систем реального часу може бути дуже велика?

1.10.2.7 Які фактори впливають на вартість супроводу ПЗ?

1.10.2.8 Як можна суттєво знизити вартість супроводу ПЗ?

1.11 Реінжиніринг програмного забезпечення

Успадкована система. Переваги реінжинірингу. Основні етапи процесу реінжинірингу. Додаткові фактори, що обумовлюють вартість реінжинірингу [4(с. 569-584)].

1.11.1 Методичні вказівки

Успадкованою системою називають стару систему, необхідну для підтримки поточної ділової активності організації, яка поки що не може від неї відмовитися.

Реінжиніринг - це повторна реалізація успадкованої системи з метою підвищення зручності її експлуатації і супроводу. У це поняття входять різні процеси, серед яких повторне документування системи, її реорганізацію і реструктуризацію, переклад системи на одну з більш сучасних мов програмування, модифікацію і модернізацію структури і системних даних. При цьому функціональність системи і її архітектура залишаються незмінними.

З технічної точки зору реінжиніринг - це рішення "другого сорту" проблеми системної еволюції. Якщо врахувати, що архітектура системи не змінюється, то зробити централізовану систему розподіленою представляється досить складною справою. Зазвичай неможна змінити мову програмування старих систем на об'єктно-орієнтовані мови (наприклад, Java або C ++). Ці обмеження вводяться для збереження архітектури системи.

Однак з комерційної точки зору реінжиніринг часто приймається за єдиний спосіб збереження успадкованих систем в експлуатації. Інші підходи до еволюції системи або занадто дорогі, або ризиковані. Щоб зрозуміти причини такої позиції, слід розглянути проблеми, пов'язані з успадкованими системами.

Програмних систем настільки багато, що говорити про повну заміну або радикальну реструктуризацію їх в більшості організацій не доводиться. Супровід старих систем дійсно коштує дорого, проте реінжиніринг може продовжити час їх існування. Реінжиніринг систем вигідний в тому випадку, якщо система має певну комерційну цінність, але дорога в супроводі. За допомогою реінжинірингу вдосконалюється системна структура, створюється нова документація та полегшується супровід системи.

У порівнянні з більш радикальними підходами до вдосконалення систем реінжиніринг має дві переваги.

1. Зниження ризиків. При повторній розробці ПЗ існують великі ризики - висока ймовірність помилок в системній специфікації і виникнення проблем під час розробки системи. Реінжиніринг знижує ці ризики.

2. Зниження витрат. Собівартість реінжинірингу значно нижче, ніж розробка нового програмного забезпечення. Вважається, що реінжиніринг в чотири рази дешевше, ніж повторна розробка системи,

Основна відмінність між реінжинірингом і новою розробкою системи пов'язано зі стартовою точкою початку роботи над системою. При реінжинірингу замість написання системної специфікації "з нуля" стара система служить основою для розробки специфікації нової системи. Традиційна розробка починається з етапу створення системної специфікації, за якою слідує проектування і реалізація нової системи. Реінжиніринг ґрунтується на існуючій системі, яка розробниками вивчається і перетворюється в нову.

На рисунку 1.10 показаний можливий процес реінжинірингу. На початку цього процесу маємо успадковану систему, а в результаті - структуровану і заново скомпоновану версію тієї ж системи. Перелічимо основні етапи цього процесу.

1. Переклад вихідного коду. Конвертація програми зі старої мови програмування на сучасну версію тієї ж мови або на іншу мову.

2. Аналіз програм. Документування структури і функціональних можливостей програм на основі їх аналізу.

3. Модифікація структури програм. Аналізується і модифікується керуюча структура програм з метою зробити їх більш простими і зрозумілими.

4. Розбиття на модулі. Взаємопов'язані частини програм групуються в модулі; там, де можливо, усувається надмірність. У деяких випадках змінюється структура системи.

5. Зміна системних даних. Дані, з якими працює програма, змінюються з тим, щоб відповідати нововведенням.

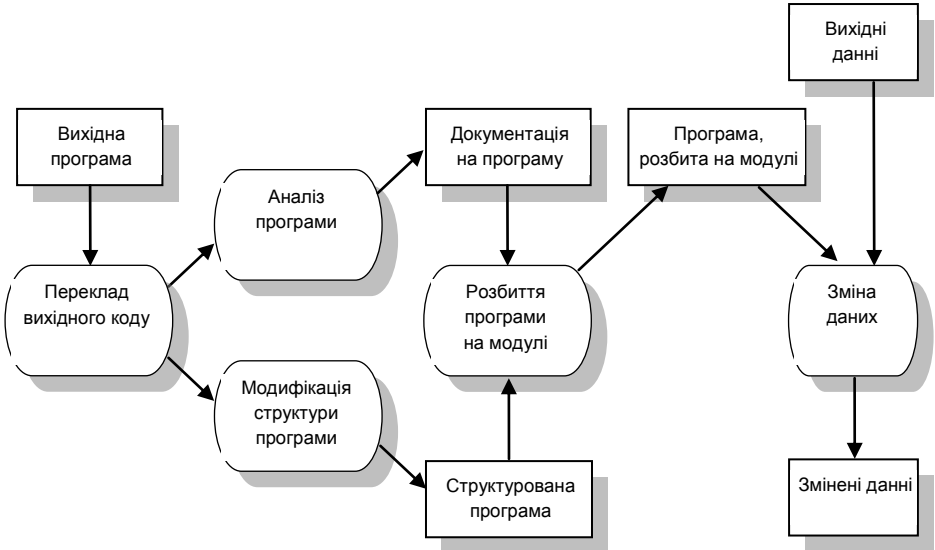


Рисунок 1.10 - Вартість реінжинірингу

При реінжинірингу програм необов'язково проходити всі стадії, показані на рис. 1.10. Наприклад, не завжди потрібно перекладати вихідний код, якщо мова програмування, на якому написана програма, все ще підтримується розробником компілятора. Якщо реінжиніринг проводиться за допомогою автоматизованих засобів, то не обов'язково відновлювати документацію на програму. Зміна системних даних необхідна, якщо в результаті реінжинірингу змінюється їх структура. Однак реструктуризація даних в процесі реінжинірингу потрібна завжди.

Вартість реінжинірингу зазвичай визначається обсягом виконаних робіт. На рис. 1.11 показано кілька різних підходів до процесу реінжинірингу і динаміка зміни вартості робіт для цих підходів.

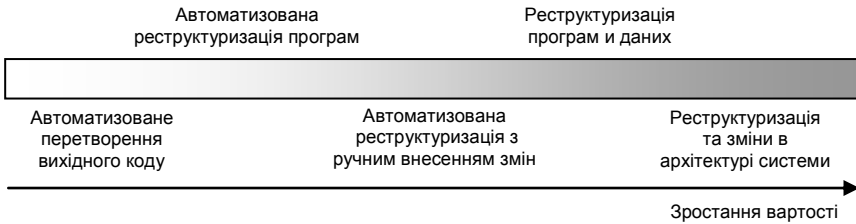


Рисунок 1.11 - Вартість реінжинірингу

Крім обсягу виконуваних робіт, є й інші фактори, що зумовлюють вартість реінжинірингу.

1. Якість програмного забезпечення, яке піддається реінжинірингу. Чим нижче якість програм і їх документації (якщо вона є в наявності), тим вище вартість реінжинірингу.

2. Наявність коштів підтримки процесу реінжинірингу. Зазвичай реінжиніринг економічно вигідний, якщо застосовуються CASE-засоби для автоматизованого внесення змін до програми.

3. Обсяг необхідного перетворення даних. Вартість процесу реінжинірингу зросте при збільшенні обсягу перетворюваних даних.

4. Наявність необхідних фахівців. Якщо персонал, який займається супроводом системи, не може виконати реінжиніринг, це також може стати причиною підвищення вартості процесу. Знову залучені фахівці витратять багато часу на вивчення системи.

Основним недоліком реінжинірингу прийнято вважати те, що з його допомогою систему можна поліпшити тільки до певної міри. Наприклад, за допомогою реінжинірингу неможливо функціонально-орієнтовану систему зробити об'єктно-орієнтованою. Основні архітектурні зміни або повну реструктуризацію програм неможливо виконати автоматично, що також збільшує вартість реінжинірингу. І, незважаючи на те, що реінжиніринг допоможе поліпшити супровід системи, все одно вона буде набагато гірше в супроводі, ніж нова, створена за допомогою сучасних методів інженерії ПЗ.

1.11.2 Запитання для самоперевірки

1.11.2.1 Що таке успадкована система?

1.11.2.2 В чому є суть реінжинірингу ПЗ?

1.11.2.3 В чому є переваги реінжинірингу ПЗ?

1.11.2.4 В чому є обмеження реінжинірингу ПЗ?

1.11.2.5 Які основні етапи процесу реінжинірингу?

1.11.2.6 Які фактори обумовлюють вартість реінжинірингу ПЗ?

1.11.2.7 Як можна знизити вартість реінжинірингу ПЗ?

1.12 Автоматизація підтримки життєвого циклу ПЗ

Структурний та об'єктно-орієнтовний підходи до проектування ПЗ. Методи структурного аналізу та проектування ПЗ. Методологія автоматизованої розробки ПЗ Rational Unified Process (RUP) та Unified Modeling Language (UML) [1(с. 216-293)].

1.12.1 Методичні вказівки

Під моделлю ПЗ в загальному випадку розуміється формалізований опис системи ПЗ на певному рівні абстракції. Кожна модель визначає конкретний аспект системи, використовує набір діаграм і документів заданого формату, а також відображає точку зору і є об'єктом діяльності різних людей з конкретними інтересами, ролями або завданнями.

Графічні (візуальні) моделі являють собою засоби для візуалізації, опису, проектування та документування архітектури системи. Хороші моделі є основою взаємодії учасників проекту та гарантують коректність архітектури. Оскільки складність систем підвищується, важливо мати у своєму розпорядженні хороші методи моделювання. Хоча є багато інших факторів, від яких залежить успіх проекту, але наявність суворого стандарту мови моделювання є дуже суттєвою.

Склад моделей, що використовуються в кожному конкретному проекті, і ступінь їх детальності в загальному випадку залежать від наступних факторів:

- складності проектованої системи;
- необхідної повноти її опису;
- знань і навичок учасників проекту;
- часу, відведеного на проектування.

Візуальне моделювання справило великий вплив на розвиток технологій створення ПЗ взагалі і CASE-засобів зокрема. Поняття CASE (Computer Aided Software Engineering) використовується в даний час в досить широкому сенсі. Первісне значення цього поняття, що обмежене тільки завданнями автоматизації розробки ПЗ, в даний

час набуло нового змісту, що охоплює більшість процесів життєвого циклу ПЗ.

CASE-технологія являє собою сукупність методів проектування ПЗ, а також набір інструментальних засобів, що дозволяють в наочній формі моделювати предметну область, аналізувати цю модель на всіх стадіях розробки і супроводу ПЗ і розробляти програми відповідно до інформаційних потреб користувачів. Більшість існуючих CASE-засобів заснована на методах структурного або об'єктно-орієнтованого аналізу і проектування, що використовують специфікації у вигляді діаграм або текстів для опису зовнішніх вимог, зв'язків між моделями системи, динаміки поведінки системи та архітектури програмних засобів.

У структурному аналізі і проектуванні використовуються різні моделі що описують:

1. Функціональну структуру системи;
2. Послідовність виконуваних дій;
3. Передачу інформації між функціональними процесами;
4. Відносини між даними.

Найбільш поширеними моделями перших трьох груп є:

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

- модель IDEF3;

- DFD (Data Flow Diagrams) - діаграми потоків даних.

Концептуальною основою об'єктно-орієнтованого аналізу і проектування ПЗ (ООАП) є об'єктна модель. Її основні принципи (абстрагування, інкапсуляція, модульність і ієрархія) і поняття (об'єкт, клас, атрибут, операція, інтерфейс і ін.) найбільш чітко сформульовані Граді Бучем в його фундаментальних роботах.

Більшість сучасних методів ООАП засновані на використанні мови UML. Уніфікована мова моделювання UML (Unified Modeling Language) представляє собою мову для визначення, уявлення, проектування та документування програмних систем, організаційно-економічних систем, технічних систем та інших систем різної природи. UML містить стандартний набір діаграм і нотацій найрізноманітніших видів.

Стандарт UML містить наступний набір діаграм:

- Структурні (structural) моделі:

- діаграми класів (class diagrams) - для моделювання статичної структури класів системи і зв'язків між ними;
- діаграми компонентів (component diagrams) - для моделювання ієрархії компонентів (підсистем) системи;
- діаграми розміщення (deployment diagrams) - для моделювання фізичної архітектури системи.
- Моделі поведінки (behavioral):
 - діаграми варіантів використання (use case diagrams) - для моделювання функціональних вимог до системи (у вигляді сценаріїв взаємодії користувачів з системою);
 - діаграми взаємодії (interaction diagrams):
 - діаграми послідовності (sequence diagrams) і кооперативні діаграми (collaboration diagrams) - для моделювання процесу обміну повідомленнями між об'єктами;
 - діаграми станів (statechart diagrams) - для моделювання поведінки об'єктів системи при переході з одного стану в інший;
 - діаграми діяльності (activity diagrams) - для моделювання поведінки системи в рамках різних варіантів використання, або потоків управління.

UML має механізми розширення призначені для того, щоб розробники могли адаптувати мову моделювання до своїх конкретних потреб, не змінюючи при цьому його метамодель. Наявність механізмів розширення принципово відрізняє UML від таких засобів моделювання, як IDEF0, IDEF1X, IDEF3, DFD та ERM. Перераховані мови моделювання можна визначити як сильно типізовані (по аналогії з мовами програмування), оскільки вони не допускають довільній інтерпретації семантики елементів моделей. UML, допускаючи таку інтерпретацію (в основному за рахунок стереотипів), є слабо типізованою мовою.

1.12.2 Запитання для самоперевірки

- 1.12.2.1 В чому відмінність структурного та об'єктно-орієнтовного підходів до проектування ПЗ?
- 1.12.2.2 Що представляє собою модель ПЗ ?
- 1.12.2.3 В чому переваги візуального моделювання ПЗ?
- 1.12.2.4 Що таке CASE-технологія?

1.12.2.5 Назвіть основні методи структурного аналізу та проектування ПЗ?

1.12.2.6 Що таке UML та в чому його призначення?

1.12.2.7 Назвіть основні структурні моделі UML?

1.12.2.8 Назвіть основні моделі поведінки UML?

2 ПЕРЕЛІК ЛАБОРАТОРНИХ РОБІТ

- 2.1 Створення діаграми варіантів використання.
- 2.2 Створення діаграм взаємодії.
- 2.3 Створення діаграм класів.
- 2.4 Створення діаграм послідовностей (з урахуванням нових вимог).
- 2.5 Доповнення діаграми класів (додавання зв'язку між класами).
- 2.6 Створення діаграми станів.
- 2.7 Створення діаграм компонентів.
- 2.8 Створення діаграм розміщення.
- 2.9 Генерація коду C++.
- 2.10 Проектування власної системи.

3 КОНТРОЛЬНІ ЗАВДАННЯ

3.1 Загальні вказівки

По даній дисципліні виконується 1 контрольна робота.

Контрольна робота базується на вивченні розділів 1.3 - 1.9 робочої програми дисципліни, та виконанні лабораторних робіт №1-9.

3.2 Завдання на контрольну роботу

У контрольній роботі необхідно розробити проектну та експлуатаційну документацію, що відповідають основним стадіям розробки запропонованої викладачем програмної системи (ПС), і в сукупності відбивають повний цикл її проектування:

3.2.1 Розробити технічне завдання на проектувану систему

- Виконати дослідження та аналіз вимог до проектуваного програмного виробу.
- Класифікувати вимоги по розділах ТЗ.
- Визначити математичні методи, що будуть застосовані на подальших стадіях проектування (привести їх у відповідних розділах ТЗ).
- Розробити ТЗ на створення програмного виробу відповідно ДЕРЖСТАНДАРТУ.

3.2.2 Розробити пояснювальну записку до технічного проекту на проектувану систему

- Виконати розробку етапів технічного проекту на створювану програмну систему.
- Класифікувати результати розробок по розділах пояснювальної записки.
- Розробити пояснювальну записку до технічного проекту.

3.2.3 Виконати кодування, налагодження та опис програмного забезпечення проектованої системи

- Виконати кодування ПЗ обраною мовою програмування.
- Розробити опис ПЗ.
- Оформити текст програми.

3.2.4 Розробити програму та методику випробувань проектованої системи

- Розробити програму випробувань ПЗ, використовуючи дані, викладені в пп. 3.2.1 – 3.2.3.
- Розробити набір тестів, що забезпечують проведення випробувань програмної системи відповідно до вимог.
- Виконати тестові прогони згідно програми випробувань.
- За результатами проведення випробувань розробити документацію (програму та методику, а також протокол випробувань).

3.2.5 Розробити експлуатаційну документацію на проектувану систему

- Розробити інструкцію користувача по експлуатації (IKE) на програмну систему.

- Оформити ІКЕ згідно вимог ДСТУ та СТП.

3.3 Методичні вказівки до виконання контрольної роботи

3.3.1 Розробка технічного завдання на створення системи

На етапі досліджень основна увага розроблювача концентрується на розумінні призначення пропонованої ПЗ і вироблення до неї вимог з боку користувача.

На стадії технічного завдання виконуються наступні основні види робіт:

- збір вихідних матеріалів і постановка задачі для рішення на ЕОМ;
- вибір і обґрунтування критеріїв ефективності і якості розроблювальної програми;
- визначення принципової можливості рішення поставленої задачі;
- попередній вибір методів рішення задач;
- визначення вимог до технічних засобів;
- визначення вимог і цілей розробки ПЗ;
- визначення стадій, етапів і термінів розробки ПЗ і документації на неї;
- визначення вимог до функцій, що повинні забезпечуватися ПЗ;
- розробка часових та об'ємних характеристик програми.

Технічне завдання включає наступні розділи:

- вступ: даються найменування та коротка характеристика області застосування програмного виробу;
- підстава для розробки: указуються документи, на підставі яких ведеться розробка;
- призначення розробки: визначається функціональне та експлуатаційне призначення програмного виробу;
- вимоги до програмного виробу:

1) вимоги до функціональних характеристик (складу виконуваних функцій, організації вхідних і вихідних даних, часовим характеристикам);

2) вимоги до надійності (забезпечення стійкого функціонування, контроль вхідної і вихідної інформації і т.д.);

3) умови експлуатації (вибір типів носіїв інформації, вид обслуговування, кількість і кваліфікація персоналу);

4) вимоги до складу і параметрів технічних засобів: вказується необхідний склад технічних засобів з указівкою їхніх основних характеристик;

5) вимоги до інформаційної і програмної сумісності: повинні бути зазначені вимоги до інформаційних структур на вході і виході та методів їхнього перетворення;

6) вимоги до маркірування та упакування;

7) вимоги до транспортування та збереження;

8) спеціальні вимоги;

- техніко-економічні показники;

- стадії та етапи розробки: установлюють необхідні стадії розробки, етапи і зміст робіт (перелік програмних документів, що повинні бути розроблені, погоджені та затверджені);

- порядок приймання і контролю: указуються види **випробувань** та загальні вимоги до приймальних робіт.

3.3.2 Розробка пояснювальної записки до технічного проекту на проектувану систему

На етапі зовнішнього проектування програмного виробу визначаються: тип мови взаємодії користувачів із програмним виробом, функції користувачів, синтаксис і семантика мови взаємодії, структура і зміст інформаційних кадрів і шаблонів діалогу, структура вхідних і вихідних даних.

На етапі проектування архітектури програмний виріб розбивають на складові частини; визначають функції кожного компонента; способи взаємодії між ними; розробляють схеми керування потоками даних; визначають схеми розподілу оперативної та зовнішньої пам'яті обчислювальної системи.

На етапі проектування структур даних визначають способи представлення, збереження та перетворення вхідних, вихідних і внутрішніх даних.

На етапі проектування модульної структури здійснюється розбивка компонентів програмного виробу на модулі, визначення їхніх функцій і способів взаємодії.

На етапі проектування модулів складаються описи всіх модулів програмного виробу, що містять у собі: ім'я модуля; функції, виконувані модулем; опис вхідних і вихідних параметрів, їхніх

форматів, діапазонів припустимих значень та ін.; функціонального взаємозв'язку між вхідними і вихідними параметрами; подій, зовнішніх стосовно модуля (друк повідомлень, робота з файлами і т.д.); форму звертання до модуля; список модулів, що викликають даний, і модулів, що викликає даний модуль; опис алгоритму модуля.

Для представлення загальної структури ПЗ програмними модулями застосовується методика спадного проектування.

Проектування ПЗ модулів здійснюється методами структурного проектування.

Документування загальної структури ПЗ та опис кожного модуля програми здійснюється за методикою, подібної НПРО-технології.

Пояснювальна записка включає наступні розділи:

- вступ;
- призначення та область застосування;
- технічні характеристики, що містять наступні підрозділи: постановка задачі, опис застосовуваних математичних методів, прийнятих допущень і обмежень, опис алгоритму і функціонування програм, опис і обґрунтування вибору методів організації вхідних та вихідних даних, складу технічних і програмних засобів;
- опис результатів, одержуваних на етапах технічного проектування (вищерозглянутих);
- очікувані техніко-економічні показники.

3.3.3 Кодування, налагодження та опис програмного забезпечення проектованої системи

Прийнятою мовою програмування здійснюється кодування структури ПЗ, розробленої на етапі технічного проектування програмної системи.

Опис ПЗ включає наступні розділи:

- загальні відомості - вказуються найменування й позначення програми; перелік програм, необхідних для її функціонування; мови програмування, на яких написана програма;
- функціональне призначення - даються класи розв'язуваних задач і відомості про функціональні обмеження на їхнє застосування;
- вхідні дані - описуються характер, організація і попередня підготовка вхідних даних, формат і спосіб їхнього кодування;

- вихідні дані - приводяться ті ж відомості, що й у попередньому розділі для вихідних даних;

- опис логічної структури містить розділи: використовувані методи, структуру програми з описом функцій складових частин і зв'язків між ними, структура й організація даних, використовуваних у програмі, алгоритм програми, зв'язок програми з іншими програмами;

- виклик і завантаження - указуються спосіб виклику програми з відповідного носія даних, вхідні точки в програму, відомості про використання оперативної пам'яті та обсяг програми;

- використовувані технічні засоби - даються типи ЕОМ і пристроїв, що застосовуються при роботі програми.

Текст програми містить символічний запис програми мовою програмування з коментарями, що відбивають структуру та призначення частин програми.

3.3.4 Розробка програми та методики випробувань проектованої системи

Для визначення правильності роботи ЗД при проведенні **випробувань** використовують різні критерії, найбільш важливі з яких:

- кожен оператор повинний бути виконаний хоча б один раз для заданого набору тестових даних і програма повинна видати правильні результати;

- кожна гілка програми повинна бути апробована з одержанням правильних результатів;

- кожен шлях у ПЗ повинний бути випробуваний хоча б один раз і програма повинна видати правильні результати;

- для кожної специфікації програми необхідно мати дані, щоб установити правильність її реалізації ПЗ;

- результати правильного тестування при прогоні тесту повинні бути відомі до початку його виконання, тому що, у протилежному випадку аналіз результатів займає багато часу і мало ймовірно, що навіть основні частини програми будуть перевірені.

З великої кількості **випробувань** ПЗ, проведених на всіх стадіях його життєвого циклу, у рамках дійсної роботи підлягають виконанню наступні категорії іспитів:

- демонстрація в дії;

- повна функціональна перевірка;

- надійність.

Під час демонстрації спеціально підібрані тести, що забезпечують бажаний результат для підтвердження всім зацікавленим особам, що виріб досяг визначеного рівня завершеності.

Ціль повної функціональної перевірки - показати, що виріб володіє усіма функціональними можливостями, зазначеними в зовнішній специфікації і, працює правильно.

Під час надійносних випробувань програмний виріб ставиться в умови, що дозволяють оцінити його здатність до стійкої роботи або відновленню після відмовлень. У ході цих **випробувань** навмисно вносяться штучно створені помилки і перевіряють усі відновлені процедури.

Програма та методика включає наступні розділи:

- об'єкт **випробувань** - дається найменування, область застосування та позначення випробовуваної програми;

- ціль **випробувань** - вказується мета проведення **випробувань**: демонстрація в дії; повна функціональна перевірка; доказ надійності ПЗ;

- склад пред'являемої документації - приводиться перелік документації, що пред'являється при проведенні **випробувань**: технічне завдання; пояснювальна записка до технічного проекту; опис програми;

- технічні вимоги - включають підрозділи:

- вимоги до програмної документації;

- технічні характеристики;

- інформаційна і програмна сумісність;

- порядок проведення **випробувань** - вказуються:

- послідовність **випробувань**;

- склад, структура технічних і програмних засобів, на яких будуть проводитися **випробування**;

- методи **випробувань** - проводиться опис використовуваних методів, а також результати проведення **випробувань**.

3.3.5 Розробка експлуатаційної документації не проектовану систему

Розробка експлуатаційної документації є завершальним етапом робочого проектування. Комплект ЕД, у загальному випадку, містить:

- відомість експлуатаційних документів;
- формуляр;
- опис застосування;
- керівництво програміста;
- керівництво системного програміста;
- опис мови;
- керівництво оператора.

З огляду на специфіку розробленої програмної системи, що полягає в учбово-методичній її спрямованості, у рамках дійсної роботи студентами розробляється один експлуатаційний документ "Інструкція користувача по експлуатації програмної системи (ЕПС)".

ЕПС узагальнює в єдиному документі основний зміст експлуатаційних документів і містить наступні основні розділи:

- призначення програми - указується призначення, можливості програми, її основні характеристики, обмеження, що накладаються на область застосування програми;
- умови застосування програми - вказуються умови, що необхідні для її виконання (обсяг оперативної пам'яті, вимоги до складу периферійних пристроїв, вимоги до системного програмного забезпечення та ін.);
- звертання до програми - описуються процедури виклику програми, способи передачі керування і параметрів;
- опис мови взаємодії - описуються синтаксис і семантика базових і складених елементів мови, оператори мови, засобу обміну даними;
- повідомлення, що видаються - вказуються тексти повідомлень, що видаються користувачу в ході запуску, виконання та закінчення роботи програми, опис їхнього змісту і дій, які необхідно почати по цих повідомленнях;
- опис бази даних (БД), з якою взаємодіє програмна система - вказується структура усіх файлів БД, позначення, призначення та розмірність полів, інформаційні потоки між змістом БД і програмою;
- додаткові розділи за розсудом студента.

ЛІТЕРАТУРА

1. Орлов, С. А. Программная инженерия: технологии разработки программного обеспечения. Учебник для вузов / С. А. Орлов - 5-е изд., обновл. и доп. – СПб.: Питер, 2016. - 640 с.
2. Лаврищева, Е.М. Software Engineering компьютерных систем. Парадигмы, технологии и CASE- средства программирования/ Е. М. Лаврищева. – К.: Наук. думка, 2013. – 283 с.
3. Орлов С.А. Технологии разработки программного обеспечения. Разработка сложных программных систем: Учеб. пособие.- 2-е изд.- СПб.: Питер, 2003.- 480 с.
4. Соммервил Иан. Инженерия программного обеспечения, 6-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2002.- 624 с.
5. Трофимов С.А. CASE-технологии: практическая работа в Rational Rose. Изд. 2-е. – М.: Бином-Пресс, 2002. – 288 с.
6. Кармайкл Э. Быстрая и качественная разработка программного обеспечения/ Э.Кармайкл, Д.Хейвуд.- Пер. с англ.- М.: Вильямс, 2003.- 400с.
7. Новиков Ф.А. Моделирование на UML. Теория, практика, видеокурс. / Ф.А. Новиков, Д.Ю.Иванов. – СПб.: Наука и техника, 2010. – 640 с.
8. Коберн А. Современные методы описания функциональных требований к системам. / А. Коберн. – М.:Лори, 2011. – 288 с.
9. Методичні вказівки до виконання лабораторних робіт з дисципліни "програмне забезпечення комп'ютерних систем" для студентів напряму 123 "Комп'ютерна інженерія" спеціалізацій (освітніх програм) "Комп'ютерні системи та мережі", "Спеціалізовані комп'ютерні системи" усіх форм навчання/ Укл.: К.М. Касьян, М.М. Касьян.- Запоріжжя: ЗНТУ, 2017. – 70 с.