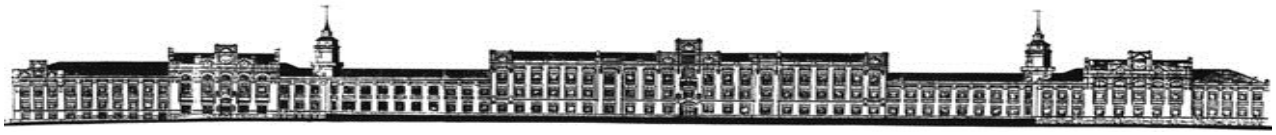


Національний технічний університет України

“Київський Політехнічний Інститут”

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ТЕХНІЧНОЇ КІБЕРНЕТИКИ

ЕЛЕКТРОННИЙ КОНСПЕКТ ЛЕКЦІЙ

З ДИСЦИПЛІНИ

“ЕМПІРИЧНІ МЕТОДИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ”

НАПРЯМ - 050103 «Програмна інженерія»

Київ - 2015

ЗМІСТ

ВСТУП

Емпіричні дослідження – це спостереження і дослідження конкретних явищ, експеримент, а також узагальнення, класифікація та опис результатів дослідження і експерименту, впровадження їх у практичну діяльність людей.

Емпіричні дослідження використовуються для відповіді на емпіричні питання, які повинні бути точно визначені згідно з даними. Як правило, дослідник має певні теорії на тему, з якої ведеться дослідження. На основі цієї теорії пропонуються певні припущення або гіпотези. З цих гіпотез робляться прогнозування конкретних подій. Ці прогнозування можуть бути перевірені відповідними експериментами. Залежно від результатів експерименту, теорії, на яких гіпотези та прогнози були засновані, будуть підтверджуватися чи спростовуватися.

1 Емпіричні методи в наукових дослідженнях

Точний аналіз даних з використанням стандартних статистичних методів у наукових дослідженнях має вирішальне значення для визначення обґрунтованості емпіричних досліджень. Статистичні формули, такі як регресія, коефіцієнт невизначеності, Т-критерій, хі-квадрат, і різні види дисперсійного аналізу мають основне значення для формування логічних, обґрунтованих висновків. Якщо емпіричні дані досягають значимості у відповідних статистичних формулах, гіпотеза підтверджується. Якщо ні, то підтверджується нульова гіпотеза (або, вірніше спростовується альтернативна гіпотеза).

В ідеалі, емпіричні дослідження дають емпіричні дані, які потім можуть бути проаналізовані на статистичну значущість.

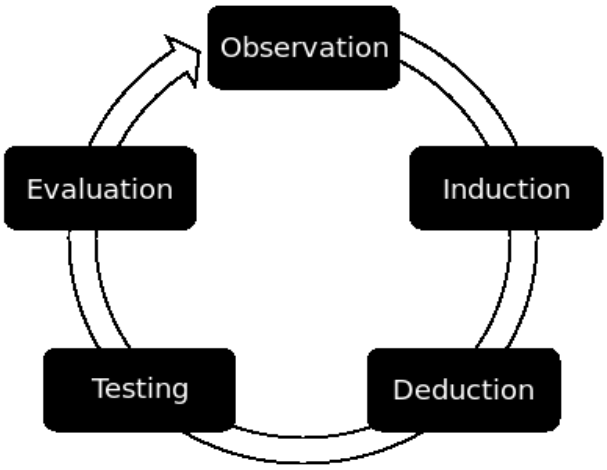


Рисунок 1.1 Основні етапи проведення емпіричного циклу

Емпіричний цикл

1) Спостереження (англ. *Observation*): збір та групування емпіричних фактів, формування гіпотези.
2) Індукція (англ. *Induction*): розробка гіпотез.
3) Дедукція (англ. *Deduction*): дедукція (виведення) послідовності гіпотез, які перевіряються прогнозуванням.
4) Перевірка (англ. *Testing*): перевірка гіпотези з нового емпіричного матеріалу.
5) Оцінка (англ. *Evaluation*): оцінка результатів перевірки.

Програмна інженерія — це застосування системного, вимірюваного підходу до розробки, використання та супроводу програмного забезпечення, та дослідження цих підходів, тобто застосування принципів інженерії до програмного забезпечення. Вперше термін «програмна інженерія (англ. software engineering)» був використаний в 1968 році на конференції з програмної інженерії, що була організована NATO.

Програмна інженерія може бути розділена на такі дисципліни:

- Вимоги: виявлення, аналіз, специфікація, перевірка вимог.
- Проектування: процес визначення архітектури, складу компонентів, інтерфейсів та інших характеристик до системи.

- Конструювання: кодування, модульне та інтеграційне тестування, відлагодження.
- Тестування: перевірка поведінки системи на відповідність до специфікації, пошук дефектів.
- Супровід програмного забезпечення. Супровід програмного забезпечення: поліпшення, оптимізація системи та процесів роботи з нею після вводу до експлуатацію.
- Конфігураційне керування: систематизує зміни до системи, що роблять розробники в процесі розробки та супровід. Попереджують небажані та непередбачені ефекти.
- Менеджмент: застосування методів та практик менеджменту для керування учасниками процесу розробки ПЗ.
- Цикл розробки ПЗ: визначення, реалізація, оцінювання, вимірювання, керування та покращення циклу розробки ПЗ як такого.
- Інструменти комп'ютерних наук: різні комп'ютерні системи що допомагають та дозволяють проводити процес розробки.
- Якість програмного забезпечення: відповідність програмного продукту вимогам.

Процес конструювання ПЗ складається з послідовності кроків, які використовують методи, утиліти і процедури. В програмотехніці існує декілька моделей, вибір тієї або іншої з них залежить перш за все від типу проекту і галузі його використання.

Застосування парадигм технології конструювання ПЗ гарантує систематичний, впорядкований підхід до промислової розробки, використання і супроводу ПЗ. Фактично, парадигми вносять в процес створення ПЗ необхідні організуючі інженерні початки.

Модель процесу конструювання – це спрощений опис процесу,

представлений з деякої точки зору, хоча моделі завжди є спрощенням. Деякі типи моделей процесу:

- Модель технологічного процесу – вказує послідовність дій, поряд зі входами, виходами і залежностями. Дії в цій моделі представляють собою дії людей.
- Модель потоків даних – представляє процес у вигляді набору дій, кожна з яких виконує деяке перетворення даних. В цій моделі дії можуть бути більш низького рівня, ніж в попередній моделі.
- Модель роль-дія – показує ролі людей, які беруть участь в процесі, а також дії, за які вони відповідають.

Існує декілька традиційних **моделей розробки програмного забезпечення:**

- Водоспадна модель;
- Еволюційний розвиток;
- Формальні перетворення – базується на створенні математичних моделей системи і їх подальшому перетворенні в програмні системи.
- Збір із повторно використовуваних компонентів – передбачає, що частини системи вже існують, і процес концентрується на інтеграції, а не на розробці.

Виділяють чотири основні **фази процесу конструювання ПЗ:**

2 Створення специфікації ПЗ – виявлення функцій системи й обмежень на розробку;

3 Розробка ПЗ – виробництво програмної системи;

4 Тестування ПЗ – перевірка того, що замовник хоче саме те, що написано в специфікації, і система відповідає специфікації;

5 Розвиток або еволюція ПЗ – зміна ПЗ у відповідь на зміни зовнішніх вимог.

2 Об’єкти дослідження емпіричними засобами програмної інженерії

2.1 Життєвий цикл програмного забезпечення

Світовий досвід розробки ПЗ дозволив виділити декілька загальноприйнятих моделей створення складних програмних систем. Ці моделі призначені для встановлення чіткої регламентації етапів і змісту робіт на кожному етапі, методів і процедур виконання самих робіт, складу і змісту розроблюваної документації. Чіткі моделі дозволяють істотно підвищити ефективність процесу розробки складних програмних комплексів, оптимально організувати управління розробкою, планувати і контролювати термінами виконання окремих етапів, правильно розподілити роботи в колективі розробників. У результаті вдається помітно знизити витрати на розробку програмного виробу і підвищити якість продукту.

Одним з базових понять методології проектування інформаційної системи є поняття життєвого циклу її ПЗ. Життєвий цикл ПЗ (ЖЦПЗ) – це безперервний процес, який починається з моменту прийняття рішення про необхідність створення ПЗ і закінчується в момент його повного вилучення з експлуатації.

Основним нормативним документом, який регламентує життєвий цикл, є міжнародний стандарт ISO/IEC 12207. Він визначає його структуру, містить процеси, дії і задачі, які повинні бути виконані під час створення ПЗ. Структура життєвого циклу за стандартом ISO/IEC 12207 базується на трьох групах процесів:

- Основні процеси (придбання, постачання, розробка, експлуатація, супровід);
- Допоміжні процеси (документування, управління конфігурацією, забезпечення якості, перевірка, атестація);
- Організаційні процеси (управління проектами, створення інфраструктури проекту, визначення, оцінка і покращення самого життєвого циклу, навчання).

Розробка включає в себе всі роботи по створенню ПЗ і його компонентів відповідно до заданих вимог, включаючи оформлення проектної і експлуатаційної документації, підготовку матеріалів, необхідних для перевірки працездатності і якості програмних продуктів, матеріалів, необхідної для організації навчання персоналу і т.д. Розробка ПЗ включає в себе, як правило, аналіз, проектування і реалізацію (програмування).

Експлуатація включає в себе роботи по впровадженню компонентів ПЗ в експлуатацію, в тому числі конфігурацію бази даних, і робочих місць користувачів, забезпечення експлуатаційною документацією, проведення навчання персоналу, і безпосередньо експлуатацію, модифікацію ПЗ, підготовку пропозицій до вдосконалення, розвитку і модернізації системи.

Управління проектом пов’язане з питаннями планування і організації робіт, створення колективів розробників і контролю за термінами і якістю виконуваних робіт. Технічне і організаційне забезпечення проекту включає вибір методів і інструментальних засобів для реалізації проекту, визначення методів опису проміжних станів розробки, розробку методів і засобів випробувань ПЗ, навчання персоналу. Забезпечення якості проекту пов’язане з проблемами перевірки і тестування ПЗ. Перевірка – це процес визначення того, чи відповідає поточний стан розробки, досягнутий на даному етапі, вимогам цього етапу. Перевірка дозволяє оцінити відповідність параметрів розробки до початкових вимог. Перевірка частково співпадає з тестуванням, яке пов’язане з ідентифікацією відмінностей між дійсними і очікуваними результатами і оцінкою відповідності характеристик початковим вимогам. У процесі реалізації проекту важливе місце займають питання ідентифікації, опису і контролю конфігурації окремих компонентів і всієї системи загалом.

Управління конфігурацією є одним з допоміжних процесів, який підтримує основні процеси життєвого циклу ПЗ, передусім процеси, розробки і супроводу ПЗ. При створенні проектів складних інформаційних систем, які складаються з багатьох компонентів, кожен з яких може мати версії, виникає

проблема обліку їх зв’язків, створення уніфікованої структури і забезпечення розвитку всієї системи. Управління конфігурацією дозволяє організувати, систематично враховувати і контролювати внесення змін в ПЗ на всіх стадіях життєвого циклу.

Кожний процес характеризується певними задачами і методами їх вирішення, початковими даними, отриманими на попередньому етапі, і результатами. Результатами аналізу, зокрема, є функціональні моделі, інформаційні моделі і відповідні їм діаграми.

2.2 Процеси тестування показників програмних засобів

Тестування продуктивності в програмній інженерії – це тестування, яке проводиться з ціллю визначення, як швидко працює програма або її частина під деяким навантаженням. Тестування продуктивності намагається враховувати продуктивність на стадії.

В тестуванні продуктивності розрізняють наступні **напрямки:**

• навантажувальне (load)

• стресове (stress)

• тестування стабільності (endurance or soak or stability)

• конфігураційне (configuration)

Навантажувальне тестування – це просто форма тестування продуктивності. Воно зазвичай проводиться для того, щоб оцінити поведінку програми(додатка) із заданим очікуваним навантаженням. Цим навантаженням може бути, наприклад, кількість користувачів, які будуть одночасно працювати з програмою. Такий вид тестування дозволяє отримати час відгуку всіх найважливіших бізнес-транзакцій.

Стресове тестування зазвичай використовується для встановлення границь пропускної здатності програми. Цей тип тестування проводиться для визначення надійності системи під час екстремальних або непропорційних

навантаженнях і відповідає на питання про достатню продуктивність системи у випадку, якщо поточне навантаження значно перевищить очікуваний максимум.

Тестування стабільності проводиться з метою переконатися в тому, що програма витримає очікуване навантаження протягом тривалого часу. При про веденні цього виду тестування здійснюється спостереження за споживанням програмою пам'яті, щоб виявити потенційні втрати. Крім цього, таке тестування виявляє деградацію продуктивності, що виражається в зниженні швидкості обробки інформації та збільшенням часу відгуку програми після тривалої роботи порівняно з початком тесту.

Конфігураційне тестування — це ще один вид традиційного тестування продуктивності. В цьому випадку замість того щоб тестувати продуктивність системи з точки зору створеного навантаження, тестується ефект впливу на продуктивність змін в конфігурації. Вдалим прикладом такого тестування можуть бути експерименти з різними методами балансування навантаження. Конфігураційне тестування може бути поєднано з навантажувальним, стресовим або тестування стабільності.

В загальних випадках тестування продуктивності може слугувати різним цілям:

- з метою демонстрації того, що система задовольняє продуктивність;
- з метою визначення продуктивність якої з двох або більше систем найкраща;
- з метою визначення, який елемент навантаження або частина системи спричиняє зниження продуктивності.

Багато тестів на продуктивність робляться без спроби осмислити їх реальні цілі. Перед початком тестування завжди задати бізнес-питання: «Яку мету ми переслідуємо, тестуючи продуктивність?». Відповіді на це питання є частиною техніко-економічного обґрунтування (або business case) тестування.

Цілі можуть відрізнятися в залежності від технологій, що використовуються додатком, або його призначення, проте, вони завжди включають щось з нижченаведеного:

1) Якщо кінцевими користувачами програми вважаються користувачі, які виконують вхід в систему в будь-якій формі, то в цьому випадку вкрай бажано досягнення паралелізму. За визначенням це максимальне число паралельних працюючих користувачів додатку, підтримка якого очікується від програми в будь-який момент часу. Модель поведінки користувача може значно впливати на здатність додатку до паралельної обробки запитів, особливо якщо він включає в себе періодичний вхід і вихід з системи.
2) Якщо концепція програми не полягає в роботі з конкретними кінцевими користувачами, то переслідувана мета для продуктивності буде заснована на максимальній пропускній здатності чи числі транзакцій за одиницю часу. Хорошим прикладом у даному випадку буде перегляд веб-сторінок, наприклад, на порталі Wikipedia.

Ця концепція будується навколо часу відповіді одного вузла додатку на запит, надісланий іншим. Простим прикладом є HTTP 'GET' запит з браузера робочої станції на веб-сервер. Практично всі програми, розроблені для навантажувального тестування працюють саме за цією схемою вимірювань. Іноді доцільно ставити завдання по досягненню продуктивності часу відповіді сервера серед всіх вузлів додатку.

Час відображення – одне з найскладніших для програми для навантажувального тестування понять, так як в загальному випадку вони не використовують концепцію роботи з тим, що відбувається на окремих вузлах системи, обмежуючись тільки розпізнаванням періоду часу протягом якого немає мережевої активності. Для того, щоб заміряти час відображення, в загальному випадку потрібно включати функціональні тестові сценарії в тести продуктивності, але більшість програм для тестування продуктивності не включають в себе таку можливість.

2.3 Вимоги до продуктивності та якості ПЗ. Зовнішні і внутрішні характеристики програм

Дуже важливо деталізувати вимоги до продуктивності і документувати їх в якому-небудь плані тестування продуктивності. В ідеальному випадку це робиться на стадії розробки вимог при розробці системи, до опрацювання деталей її дизайну. Проте тестування продуктивності часто не проводиться згідно зі специфікацією, так як немає зафіксованого розуміння про максимальний час відповіді для заданого числа користувачів. Тестування продуктивності часто використовується як частина процесу профайлингу продуктивності. Його ідея полягає в тому, щоб знайти «слабку ланку» - таку частину системи, збільшивши час реакції якої, можна поліпшити загальну продуктивність системи. Визначення конкретної частини системи, що стоїть на цьому критичному шляху, іноді дуже непросте завдання, тому деякі програми для тестування включають в себе (або можуть бути додані за допомогою add-on'ів) інструменти, запущені на сервері (агенти) і спостерігають за часом виконання транзакцій, часом доступу до бази даних. Тестування продуктивності може проводитися з використанням глобальної мережі і навіть у географічно віддалених місцях, якщо враховувати той факт, що швидкість роботи мережі Інтернет залежить від місця розташування. Воно також може проводитися і локально, але в цьому випадку необхідно налаштувати мережеві маршрутизатори таким чином, щоб з'явилася затримка, присутня в усіх публічних мережах. Навантаження, що додається до системи, повинно збігатися з реальним станом справ. Так наприклад, якщо 50% користувачів системи для доступу до системи використовують мережевий канал шириною 56К, а інша половина використовує оптичний канал, то комп'ютери, що створюють тестове навантаження на систему повинні використовувати ті ж з'єднання (ідеальний варіант) або емулювати затримки вищевказаних мережевих з'єднань.

Якість програмного забезпечення — характеристика програмного забезпечення, ступінь відповідності ПЗ до вимог. При цьому вимоги можуть трактуватись по-різному, що породжує декілька незалежних визначень терміну.

Якість ПЗ – набір властивостей продукту (сервісу або програм), що характеризують його здатність задовольнити встановлені або передбачувані потреби замовника. Поняття якості має різні інтерпретації залежно від конкретної програмної системи і вимог до неї.

Якість коду може визначатись різними критеріями. Деякі з них мають значення тільки з точки зору людини. Наприклад, форматування тексту програми — неважливо для комп'ютеру, але може мати велике значення для супроводу. Багато з існуючих стандартів кодування, що визначають специфічні для мови програмування угоди та задають низку правил, мають на меті полегшити супровід ПЗ в майбутньому. Також існують інші критерії, що визначають чи «гарно» написаний код, наприклад, такі, як структурованість — ступінь логічного розділення коду на блоки.

Деякі фактори:

- Прочитність коду
- Легкість підтримки, тестування, відлагодження, виправляння помилок, рефакторингу та портування
- Низька складність коду
- Коректність обробки винятків
- Методи покращення якості коду: рефакторинг.

Стандарт ISO/IEC 9126 регламентує зовнішні і внутрішні характеристики якості. Перші відображають вимоги до функціонування програмного продукту. Для кількісного встановлення критеріїв якості, за якими буде здійснюватися перевірка і підтвердження відповідності ПЗ заданим вимогам, визначаються відповідні зовнішні вимірювані властивості (зовнішні атрибути) ПЗ, метрики (наприклад, час виконання окремих компонентів), діапазони зміни значень і моделі їх оцінки. Метрики використовуються на стадії тестування або функціонування і називаються зовнішніми метриками. Вони являють собою моделі оцінки атрибутів.

Внутрішні характеристики якості і внутрішні атрибути ПЗ використовуються для складання плану досягнення необхідних зовнішніх характеристик якості продукту. Для квантифікації внутрішніх характеристик якості застосовують внутрішні метрики, як інструмент перевірки відповідності проміжних продуктів внутрішнім вимогам до якості, які формулюються на процесах, що передують тестуванню.

Зовнішні і внутрішні характеристики якості відображають властивості самого ПЗ (працюючого або не працюючого), а також погляд замовника і розробника на таке ПЗ. Безпосереднього кінцевого користувача ПЗ цікавить експлуатаційна якість ПЗ – сукупний ефект від досягнення характеристик якості, що виміряється строком результату, а не властивістю самого ПЗ. Це поняття ширше, ніж будь-яка окрема характеристика (наприклад, зручність використання або надійність).

Моделі мають різну кількість рівнів і повністю або частково збігаються щодо набору характеристик якості. Наприклад, модель якості МакКолла на найвищому рівні має три характеристики: функціональність, модифікованість і переносність, а на нижчих рівнях моделі – 11 підхарактеристик якості і 18 критеріїв (атрибутів) якості. Стандарт ISO 9126 пропонує використовувати для опису внутрішнього та зовнішнього якості ПЗ багаторівневу модель. На верхньому рівні виділено 6 основних характеристик якості ПЗ. Кожна характеристика описується за допомогою кількох вхідних у неї атрибутів. Для кожного атрибута визначається набір метрик, що дозволяють його оцінити. Множина характеристик і атрибутів якості згідно з ISO 9126.

Визначення цих характеристик і атрибутів за стандартом ISO 9126:2001:

Функціональність (functionality). Здатність ПЗ в певних умовах вирішувати задачі, потрібні користувачам. Визначає, що саме робить ПЗ, які задачі воно вирішує.

Функціональна придатність (suitability). Здатність вирішувати потрібний набір задач.

Точність (accuracy). Здатність видавати потрібні результати.

Здатність до взаємодії (interoperability). Здатність взаємодіяти з потрібним набором інших систем.

Відповідність стандартам і правилам (compliance). Відповідність ПЗ наявним індустріальним стандартам, нормативним і законодавчим актам, іншим регулюючим нормам.

Захищеність (security). Здатність запобігати неавторизованому, тобто без вказівки особи, що намагається його здійснити, і недозволеному доступу до даних і програм.

Надійність (reliability). Здатність ПЗ підтримувати визначену працездатність у заданих умовах.

Зрілість, завершеність (maturity). Величина, зворотна частоті відмов ПЗ. Звичайно виміряється середнім часом роботи без збоїв і величиною, зворотною імовірності виникнення відмови за даний період часу.

Стійкість до відмов (fault tolerance). Здатність підтримувати заданий рівень працездатності при відмовах і порушеннях правил взаємодії з середовищем.

Здатність до відновлення (recoverability). Здатність відновлювати визначений рівень працездатності й цілісність даних після відмови, необхідні для цього час і ресурси.

Відповідність стандартам надійності (reliability compliance). Цей атрибут доданий в 2001 році.

Зручність використання (usability) або практичність. Здатність ПЗ бути зручним у навчанні та використанні, а також привабливим для користувачів.

Зрозумілість (understandability). Показник, зворотний до зусиль, які затрачаються користувачами на сприйняття основних понять ПЗ та усвідомлення їх застосовності для розв'язання своїх задач.

Зручність навчання (learnability). Показник, зворотний зусиллям, затрачуваним користувачами на навчання роботі з ПЗ.

Зручність роботи (operability). Показник, зворотний зусиллям, що вживається користувачами для розв'язання своїх задач за допомогою ПЗ.

Привабливість (attractiveness). Здатність ПЗ бути привабливим для користувачів. Цей атрибут доданий в 2001 році.

Відповідність стандартам зручності використання (usability compliance). Цей атрибут доданий в 2001 році.

Продуктивність (efficiency) або *ефективність*. Здатність ПЗ при заданих умовах забезпечувати необхідну працездатність стосовно виділюваного для цього ресурсам. Можна визначити її і як відношення одержуваних за допомогою ПЗ результатів до затрачуваних на це ресурсів усіх типів.

Часова ефективність (time behaviour). Здатність ПЗ видавати очікувані результати, а також забезпечувати передачу необхідного об'єму даних за відведений час.

Ефективність використання ресурсів (resource utilisation). Здатність вирішувати потрібні задачі з використанням визначених об'ємів ресурсів визначених видів. Маються на увазі такі ресурси, як оперативна й довгострокова пам'ять, мережні з'єднання, пристрої вводу та виводу та ін.

Відповідність стандартам продуктивності (efficiency compliance). Цей атрибут доданий в 2001 році.

Зручність супроводу (maintainability). Зручність проведення всіх видів діяльності, пов'язаних із супроводом програм.

Аналізованість (analyzability) або зручність проведення аналізу. Зручність проведення аналізу помилок, дефектів і недоліків, а також зручність аналізу необхідності змін і їх можливих наслідків.

Зручність внесення змін (changeability). Показник, зворотний

трудозатратам на виконання необхідних змін.

Стабільність (stability). Показник, зворотний ризику виникнення несподіваних ефектів при внесенні необхідних змін.

Зручність перевірки (testability). Показник, зворотний трудозатратам на проведення тестування і інших видів перевірки того, що внесені зміни привели до потрібних результатів.

Відповідність стандартам зручності супроводу (maintainability compliance). Цей атрибут доданий в 2001 році.

Переносимість (portability). Здатність ПЗ зберігати працездатність при перенесенні з одного оточення в інше, включаючи організаційні, апаратні й програмні аспекти оточення.

Адаптованість (adaptability). Здатність ПЗ пристосовуватися різним оточенням без проведення для цього дій, крім заздалегідь передбачених.

Зручність установки (installability). Здатність ПЗ бути встановленим або розгорнутим у визначеному оточенні.

Здатність до співіснування (coexistence). Здатність ПЗ співіснувати з іншими програмами у загальному оточенні, ділячи з ними ресурси.

Зручність заміни (replaceability) іншого ПЗ даним. Можливість застосування даного ПЗ замість інших програмних систем для вирішення тих же задач у певному оточенні.

Відповідність стандартам переносимості (portability compliance). Цей атрибут доданий в 2001 році.

Нами було проведено глибоку оцінку всіх властивостей програмного забезпечення таких, як зрозумілість, повнота, стислість, портованість, погоджуваність, супроводжуваність, тестованість, юзабіліті, надійність, структурованість, ефективність, безпека, зрозумілість інтерфейсу, легкість виконання операції , зрозумілість повідомлень про помилки, очікуваність

функціональності та документованість, але при у відповідності завданню курсової роботи, наведено властивості зрозумілості повідомлень про помилки та повноти.

2.4 Метрики програмного забезпечення

Метрика програмного забезпечення (англ. Software metric) - захід, що дозволяє отримати чисельне значення деякої властивості програмного забезпечення або його специфікацій.

Оскільки кількісні методи добре зарекомендували себе в інших областях, багато теоретиків і практики інформатики намагалися перенести даний підхід і в розробку програмного забезпечення. Як сказав Том ДеМарко, «ви не можете контролювати те, що не можете виміряти», перефразувавши вислів Д. І. Менделєєва - «наука починається там, де починають вимірювати».

Метрика пов'язана з якістю програмного забезпечення.

Набір використовуваних метрик включає:

1) порядок зростання (мається на увазі аналіз алгоритмів в термінах асимптотичного аналізу та О-нотації),

2) кількість рядків коду,

3) цикломатична складність,

4) аналіз функціональних точок,

5) кількість помилок на 1000 рядків коду,

6) ступінь покриття коду тестуванням,

7) покриття вимог,

8) кількість класів і інтерфейсів,

9) метрики програмного пакета від Роберта Сесіль Мартіна,

10) зв'язність.

Потенційні недоліки підходу, на які націлена критика

Серед основних недоліків, які пов’язані з проблемою широкого застосування метрик є такі:

Неетичність: Стверджується, що неетично судити про продуктивність програміста по метрикам, введеним для оцінки ефективності програмного коду. Такі відомі метрики, як кількість рядків коду і цикломатична складність, часто дають поверхове уявлення про "вдалості" вибору того чи іншого підходу при

вирішенні поставлених завдань, однак, нерідко вони розглядаються, як інструмент оцінки якості роботи розробника. Такий підхід досить часто призводить до зворотного ефекту, приводячи до появи в коді більш довгих конструкцій та надлишкових необов'язкових методів.

Заміщення «управління людьми» на «управління цифрами», яке не враховує досвід співробітників та їх інші якості

Спотворення: Процес вимірювання може бути спотворений за рахунок того, що співробітники знають про вимірюваних показниках і прагнуть оптимізувати ці показники, а не свою роботу. Наприклад, якщо кількість рядків вихідного коду є важливим показником, то програмісти будуть прагнути писати якомога більше рядків і не будуть використовувати способи спрощення коду, що скорочують кількість рядків.

Неточність: Немає метрик, які були б одночасно і значимі і досить точні. Кількість рядків коду - це просто кількість рядків, цей показник не дає уявлення про складність вирішуваної проблеми. Аналіз функціональних точок був розроблений з метою кращого вимірювання складності коду і специфікації, але він використовує особисті оцінки вимірює, тому різні люди отримають різні результати.

Основні метрики коду, що застосовують на практиці: LOC, SLOC, Джилб, Маккейб, Холстед, ООП та інші.

2.5 Приклад опису значень характеристик і оцінок. Експертне оцінювання і метрики в програмуванні

Повнота - всі необхідні частини програми повинні бути представлені і повністю реалізовані.

Ефективність - Чи видає програма зрозумілі повідомлення про помилки?

Таблиця 2.1 - Шкала оцінок параметру «Повнота»

0-3	Представлені не всі частини програми. Реалізація не функціонує.
4-6	Основні компоненти існують та деякі з них функціонують.
7-8	Представлені майже всі компоненти програми. Реалізація також всіх присутня в повному обсязі
9-10	Представлені всі компоненти. Функціонує реалізація всіх компонентів.

Таблиця 2.2 - Шкала оцінок параметру «Зрозумілість повідомлень про помилки»

-3	Програма не видає повідомлення про помилки.
-6	Програма видає повідомлення про помилки, але вони є незрозумілими.
-8	Програма видає повідомлення про помилки, але вони бувають не чіткі.
-10	Програма видає чіткі та зрозумілі повідомлення про помилки.

Таблиця 2.3. - Експертні оцінки по проектах

Назва проекту	Зрозумілість повідомлень про помилки	Повнота
amnesia	7	9
analyzer	5	10
cdk	10	10
Chekers	4	10
ConstructionProject6	9	10
datacrow	10	10
freecol	9	10
hodoku	9	9
jackson	6	8
jdec	10	5
jfreechart	10	10
jgraphx	7	10

jpcsp	9	10
jplayer	9	9
jsmooth	9	10
Lab_Work_Arch_7_app	8	9
mp3dings	5	10
MST	7	10
sc1.2interface2	4	7
Star World	4	7
xtrememp	8	9

При вимірюванні програмного коду використовувались наступні метрики:

Таблиця 2.4 - Відстежені метрики

Прямі метрики	Непрямі метрики
LOC, NOM, NOP, HDD, HIT	CC, CDISP, CINT

Опис використаних метрик

Таблиця 2.5 - Описи прямих метрик

Назва метрика	Опис метрики
LOC	Показник кількості рядків коду
NOM	Показник кількості числа методів
NOP	Показник кількості числа пакетів
HDD	Показник кількості прямих

	нащадків
HIT	Глибина дерева успадкування

Таблиця 2.6. - Опис непрямих метрик

Назва метрика	Опис метрика
CC	Число класів, в яких визначаються методи, викликанні виміряними методами
CDISP	Число класів, в яких операції викликанні вимірюваними операціями від , поділено на CINT
CINT	Число роздільних операцій викликаних виміряними операціями

При виконанні даного аналізу були використані наступні програмні засоби:

- iPlasma – дає можливість отримати значення 80-х об’єктно-орієнтованих метрик. Функціонально повний засіб для вимірювання, який вимірює метрики, які відносяться як до окремих класів, методів та пакетів, так і для проекту в цілому. Крім того, метрики виводяться не тільки в числовому вигляді, а й у графічному – у вигляді гістограми.
- Statistica – пакет для всебічного статистичного аналізу, розроблений компанією StatSoft. У пакеті STATISTICA реалізовані процедури для аналізу даних (data analysis), управління даними (data management), видобутку даних (data mining), візуалізації даних (data visualization).

3 Статистичний аналіз метрик та експертних оцінок

Метою первинного статистичного аналізу являється визначення закону розподілу випадкової величини, точніше визначення відповіді на питання „Чи є

даний закон розподілу випадкової величини нормальним?”. На етапі первинного статистичного аналізу відбувається дослідження вхідних статистичних даних. Спочатку аналізуються метрики, отримані в результаті вимірювання набору програм, далі експертні оцінки, що зробили експерти для цього ж набору програм. В ході дослідження спочатку виявляється графічний вигляд (гістограма) закону розподілу. Після побудови гістограми за її виглядом можна відсіяти частину метрик, які мають багатомодальний вигляд, так як статистичний аналіз залежностей побудований на дослідженні унімодальних законів розподілу.

Кореляційний аналіз пар „метрика – експертна оцінка”

На етапі кореляційного аналізу визначається, чи існує залежність між певними метриками та експертними оцінками, чи її немає. Якщо залежність існує, то проводиться первинна обробка даних для визначення довірчої ймовірності та виду залежності. В іншому випадку робиться висновок про відсутність залежності.

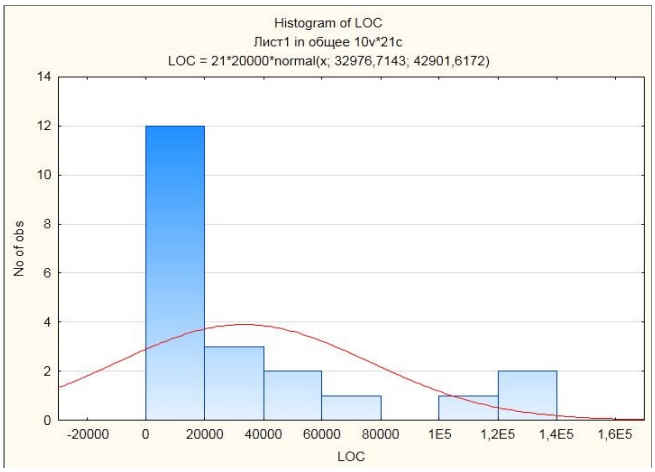
Регресійний аналіз залежних величин

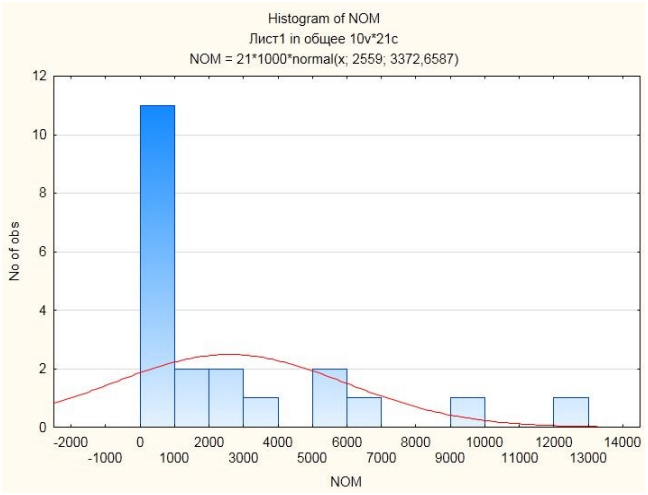
Регресійний аналіз – останній етап в дослідженні на залежність метрик та експертних оцінок. Він проводиться тільки при виконанні умови, що дисперсія залежної змінної (експертної оцінки) повинна залишатися постійною при зміні значення аргументу (метрики), тобто, спочатку визначається дисперсія експертної оцінки для кожного прийнятого значення метрики. Далі проводиться ідентифікація регресії. Вона передбачає як графічну побудову, так і аналітичні дослідження. Графічна побудова розпочинається з визначення кореляційного поля. Якщо кореляційне поле має форму еліпса, робиться висновок про лінійний регресійний зв’язок. Далі проводиться побудова лінійної регресії і її оцінка. Якщо побудовані точки кореляційного поля потрапляють у коло, то робиться висновок про відсутність залежності. Якщо ж кореляційне поле не вписується у коло чи еліпс, а має інший вигляд, то робиться висновок про нелінійну залежність в лінії регресії. Далі будуються і аналізуються найймовірніші наближені лінії регресії. Серед них вибирається найточніша

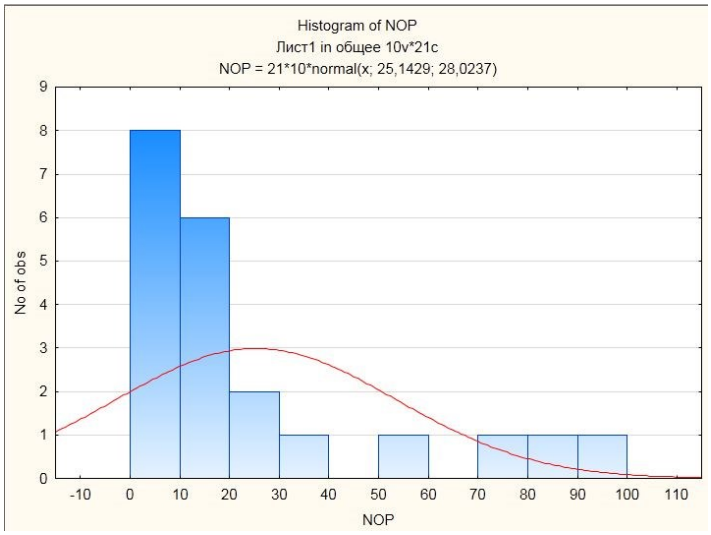
шляхом обчислення відхилення значень залежної змінної. Висновок про найточніше припущення робиться для функції, у якої відхилення найменше. Далі для нелінійної залежності проводиться лінеаризація коефіцієнтів, тобто зведення функції до лінійного вигляду.

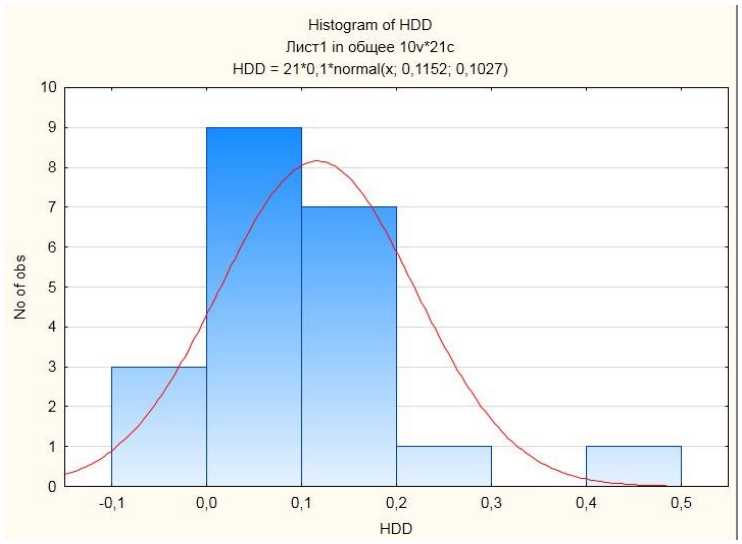
3.1 Первинний статистичний аналіз

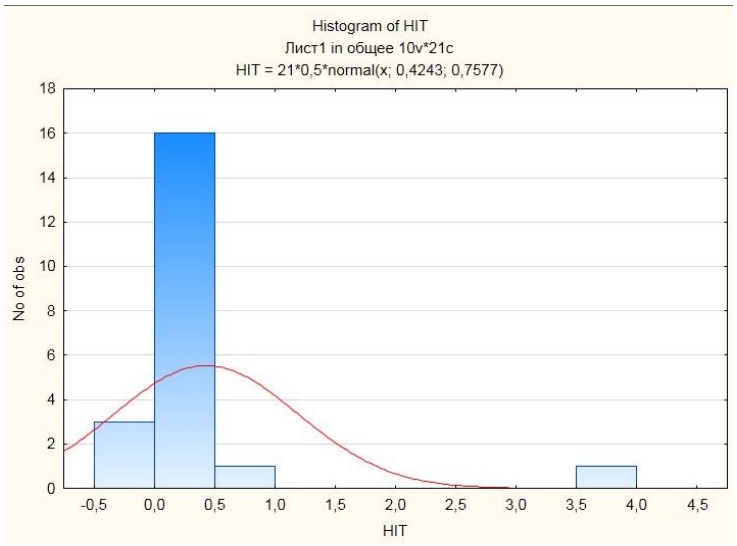
Гістограми метрик

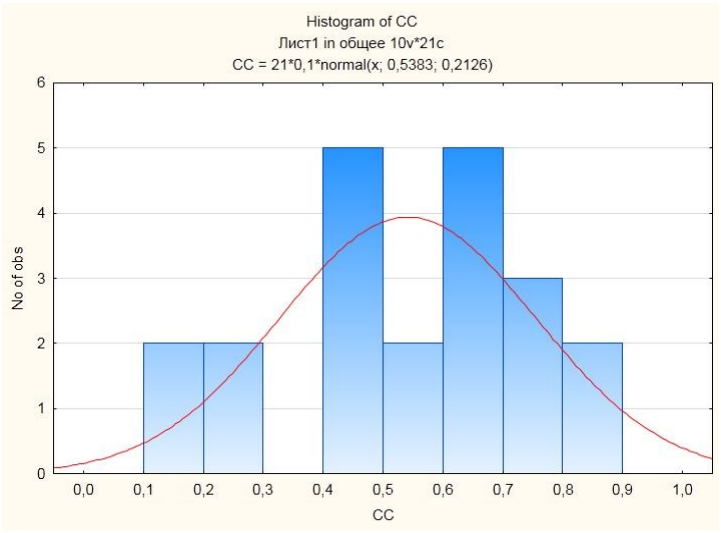


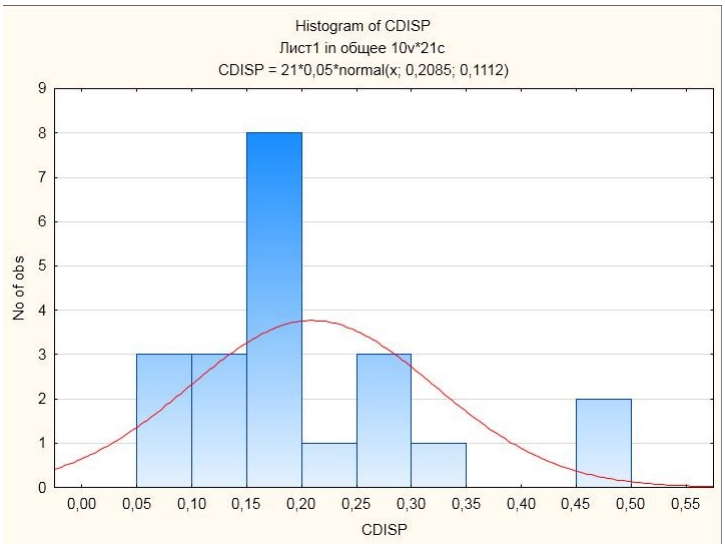


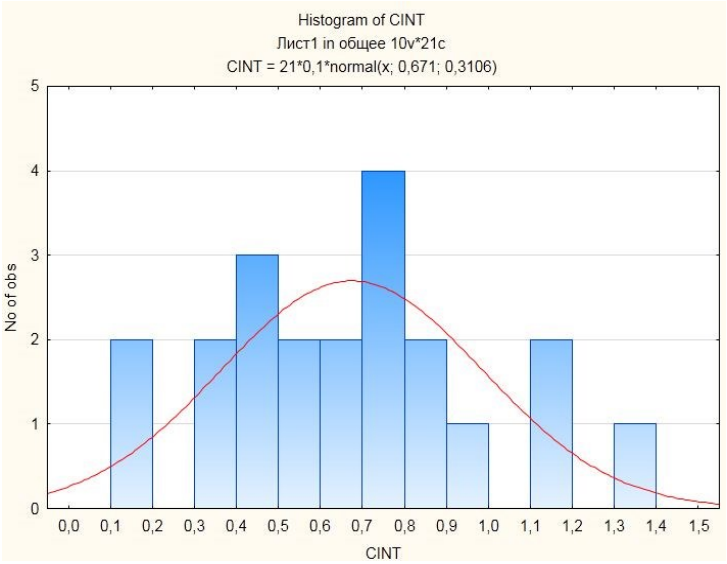

Histogram of NOP
Лист1 in общее 10v*21c
NOP = 21*10*normal(x; 25,1429; 28,0237)
No of obs
NOP
-10
0
10
20
30
40
50
60
70
80
90
100
110

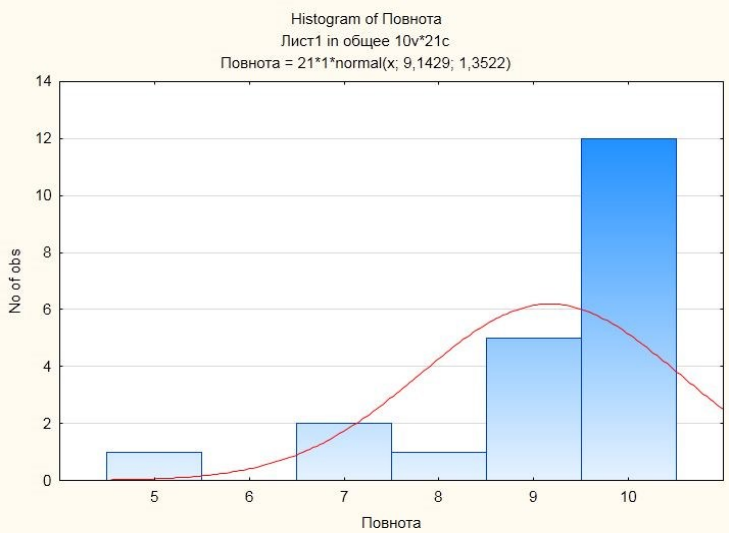

Histogram of HDD
Лист1 in общее 10v*21c
HDD = 21*0,1*normal(x; 0,1152; 0,1027)
No of obs
HDD
-0,1
0,0
0,1
0,2
0,3
0,4
0,5


Histogram of HIT
Лист1 in общее 10v*21c
HIT = 21*0,5*normal(x; 0,4243; 0,7577)
No of obs
HIT
-0,5
0,0
0,5
1,0
1,5
2,0
2,5
3,0
3,5
4,0
4,5


Histogram of CC
Лист1 in общее 10v*21c
CC = 21*0,1*normal(x; 0,5383; 0,2126)
No of obs
CC
0
1
2
3
4
5
6
0,0
0,1
0,2
0,3
0,4
0,5
0,6
0,7
0,8
0,9
1,0


Histogram of CDISP
Лист1 in общее 10v*21c
CDISP = 21*0,05*normal(x; 0,2085; 0,1112)
No of obs
CDISP
0
1
2
3
4
5
6
7
8
9
0,00
0,05
0,10
0,15
0,20
0,25
0,30
0,35
0,40
0,45
0,50
0,55


Histogram of CINT
Лист1 in общее 10v*21c
CINT = 21*0,1*normal(x; 0,671; 0,3106)
No of obs
CINT
0
1
2
3
4
5
0,0
0,1
0,2
0,3
0,4
0,5
0,6
0,7
0,8
0,9
1,0
1,1
1,2
1,3
1,4
1,5



3.2 Основні статистичні оцінки для кожної метрики

Таблиця 3.1 – Статистичні оцінки різних метрик

Variable	Descriptive Statistics (Лист1 in общее)								
	Mean	Confidence -95,000%	Confidence 95,000%	Minimum	Maximum	Variance	Std.Dev.	Skewness	Kurtosis
LOC	32976,71	13448,13	52505,30	430,0000	139880,0	1,840549E+09	42901,62	1,46221	1,11145
NOM	2559,00	1023,78	4094,22	21,0000	12105,0	1,137483E+07	3372,66	1,75257	2,60994
NOP	25,14	12,39	37,90	2,0000	92,0	7,853286E+02	28,02	1,52816	1,25711
HDD	0,12	0,07	0,16	0,0000	0,4	1,054619E-02	0,10	1,56413	3,35984
HIT	0,42	0,08	0,77	0,0000	3,6	5,741457E-01	0,76	4,21866	18,70352
CC	0,54	0,44	0,64	0,1263	0,9	4,519565E-02	0,21	-0,25616	-0,67992
CDISP	0,21	0,16	0,26	0,0806	0,5	1,235602E-02	0,11	1,32762	1,46578
CINT	0,67	0,53	0,81	0,1663	1,3	9,649217E-02	0,31	0,34187	-0,31288
Зроз	7,57	6,60	8,54	4,0000	10,0	4,557143E+00	2,13	-0,56899	-1,08205
Повнота	9,14	8,53	9,76	5,0000	10,0	1,828571E+00	1,35	-1,89378	3,45904

При проведені первинного статистичного аналізу було обчислено статичні характеристики такі, як математичне сподівання, середнє квадратичне відхилення, коефіцієнт ексцесу та асиметрії, довірчі інтервали та визначено закони розподілу.

Основними критеріями, за якими визначають належність даного закону розподілу до нормального є коефіцієнти асиметрії та ексцесу. Для нормального розподілу коефіцієнти ексцесу та асиметрії приймають значення 0. Тобто для висновку про „нормальність” розподілу потрібно, щоб всі числові характеристики потрапили в довірчі інтервали і коефіцієнти ексцесу та асиметрії наближалися до 0.

У відповідності до отриманих даних можна зробити наступні висновки:

Метрики з ненормальним розподілом:

- NOM
- LOC
- NOP
- HIT
- CDISP
- CC
- CINT
- FDP
- HDD
- Зрозумілість повідомлень про помилки
- Повнота

Ці дані будуть в подальшому використовуватись для кореляційного та регресійного аналізу.

3.3 Кореляційний аналіз

Кореляція (або коефіцієнт кореляції) є мірою залежності двох випадкових величин. При цьому, зміна однієї або кількох цих величин призводить до систематичної зміни іншої або інших величин. Математичною мірою кореляції двох випадкових величин слугує коефіцієнт кореляції. Кореляція може бути позитивною та негативною (можлива також ситуація відсутності статистичного зв'язку - наприклад, для незалежних випадкових величин). Від'ємна кореляція – кореляція, при якій збільшення однієї змінної пов'язане зі зменшенням іншої, при цьому коефіцієнт кореляції від'ємний. Додатна кореляція – кореляція, при якій збільшення однієї змінної пов'язане зі збільшенням іншої, при цьому коефіцієнт кореляції додатній.

Кореляційний аналіз – метод обробки статистичних даних, що полягає у вивченні коефіцієнту кореляції між змінними. При цьому порівнюються коефіцієнти між однією парою або множиною пар ознак для встановлення між ними статистичної взаємодії.

Мета кореляційного аналізу – забезпечити отримання деякої інформації про одну змінну за допомогою іншої змінної. В випадках, коли можливе досягнення мети, говорять, що змінні корелюють. В самому загальному вигляді сприйняття гіпотези про наявність кореляції означає, що зміна значення змінної

А відбудеться одночасно з пропорційною зміною значення B.

Кореляція відображає лише лінійну залежність величин, але не відображає їх функціональної зв’язаності.

Таблиця 3.2 – Коефіцієнти кореляцій для різних метрик

Variable	Spearman Rank Order Correlations (Лист1 in общее.stw) MD pairwise deleted Marked correlations are significant at p <,05000							
	LOC	NOM	NOP	CC	CDISP	CINT	Зроз	Повнота
LOC	1,000000	0,985714	0,866429	0,588312	-0,029870	0,616883	0,691685	0,201745
NOM	0,985714	1,000000	0,869029	0,618182	-0,035065	0,605195	0,665895	0,301892
NOP	0,866429	0,869029	1,000000	0,573286	-0,023399	0,490088	0,676473	0,337780
CC	0,588312	0,618182	0,573286	1,000000	-0,084416	0,590909	0,346504	0,463723
CDISP	-0,029870	-0,035065	-0,023399	-0,084416	1,000000	0,510390	-0,130269	0,244561
CINT	0,616883	0,605195	0,490088	0,590909	0,510390	1,000000	0,354439	0,240933
Зроз	0,691685	0,665895	0,676473	0,346504	-0,130269	0,354439	1,000000	0,210990
Повнота	0,201745	0,301892	0,337780	0,463723	0,244561	0,240933	0,210990	1,000000

При проведенні кореляційного аналізу було за допомогою пакету Statistica обчислено коефіцієнти для пар метрика-експертна оцінка.

На основі отриманих даних, які подані в таблиці 4, можна зробити наступні висновки:

- значення коефіцієнтів кореляції для пар «метрика-експертна оцінка», що зображені червоним кольором вказують на залежність між метриками та експертною оцінкою;
- значення коефіцієнтів кореляції для пар «метрика-експертна оцінка», що зображені чорним кольором не залежні між собою (мала залежність);
- коефіцієнти кореляції не є дуже великим, що дає змогу сказати, що залежність між метриками та експертними оцінками, які досліджуються в даній курсовій роботі, не є значною.

Дані, що були отримані при проведені кореляційного аналізу, будуть використані з метою проведення регресійного аналізу.

Перелік пар метрик із значимими тісними залежностями: LOC-NOM; LOC-NOP; NOM-NOP; LOC-Зрозумілість.

3.4 Регресійний аналіз

Регресія – форма зв’язку між випадковими величинами. Закон зміни математичного очікування однієї випадкової величини залежно від значень іншої. Розрізняють прямолінійну, криволінійну, ортогональну, параболічну та ін. Р., а також лінію і площину регресії.

Пари метрик, у яких був зроблений висновок про незначущість результатів відкинути. Для решти – побудували регресію.

Рисунок 3.1 – Кореляційні залежності для пари LOC-NOM:

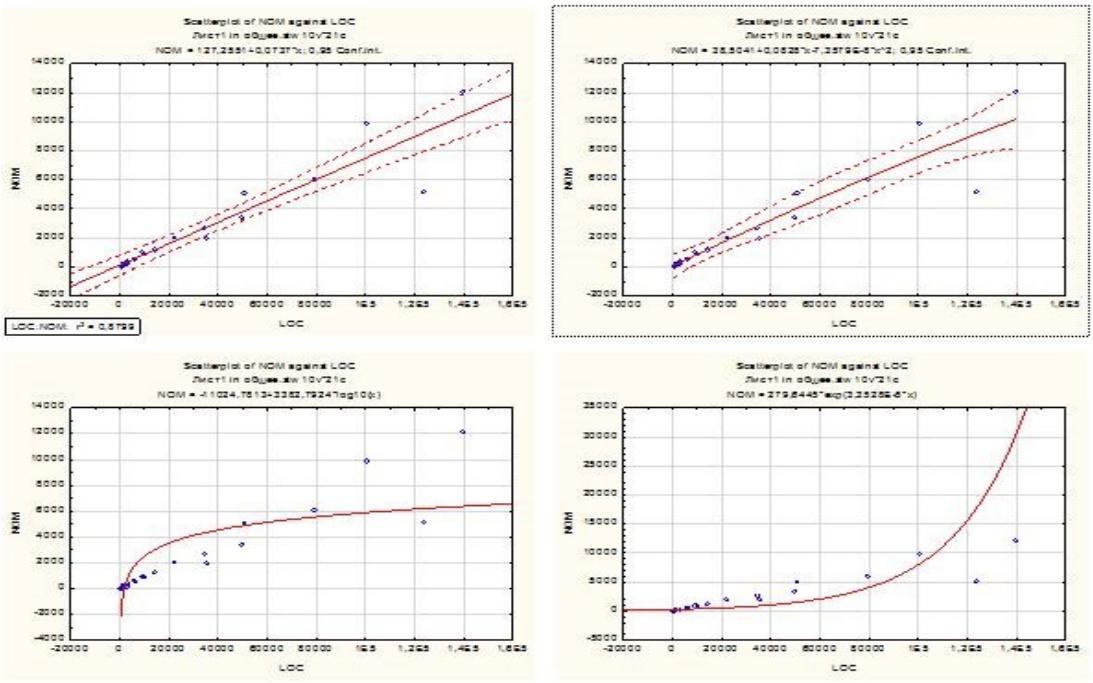


Рисунок 3.2 – Кореляційні залежності для пари LOC-NOP

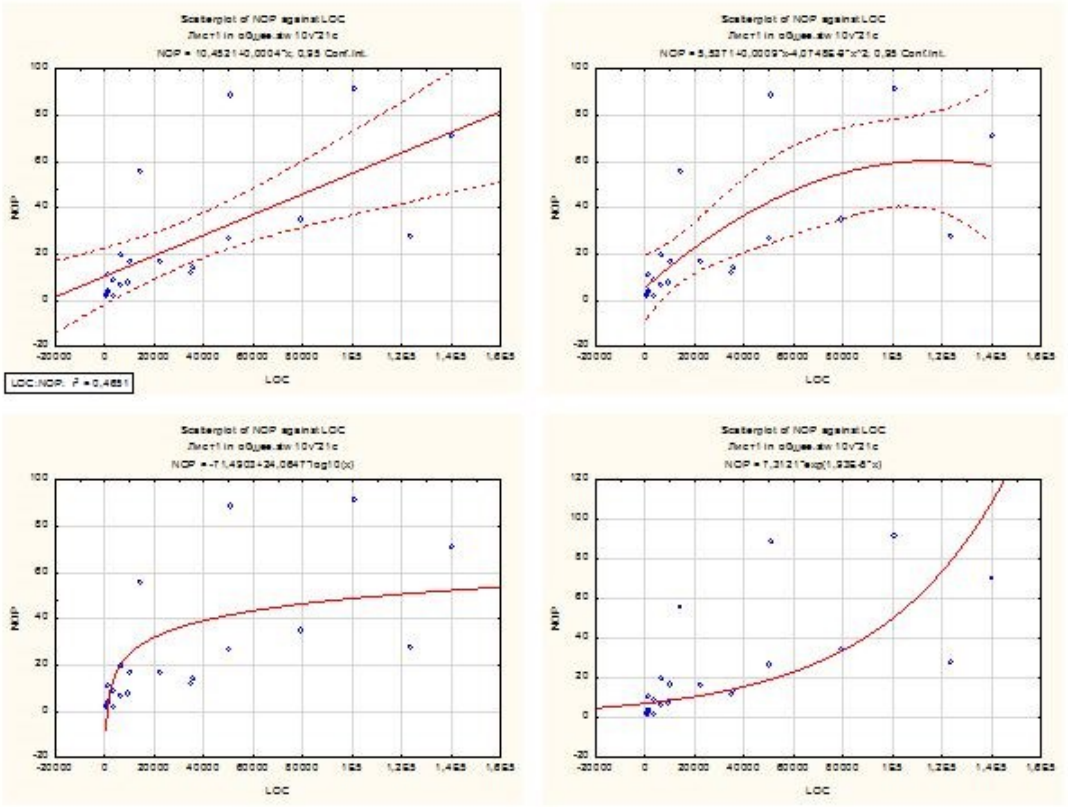
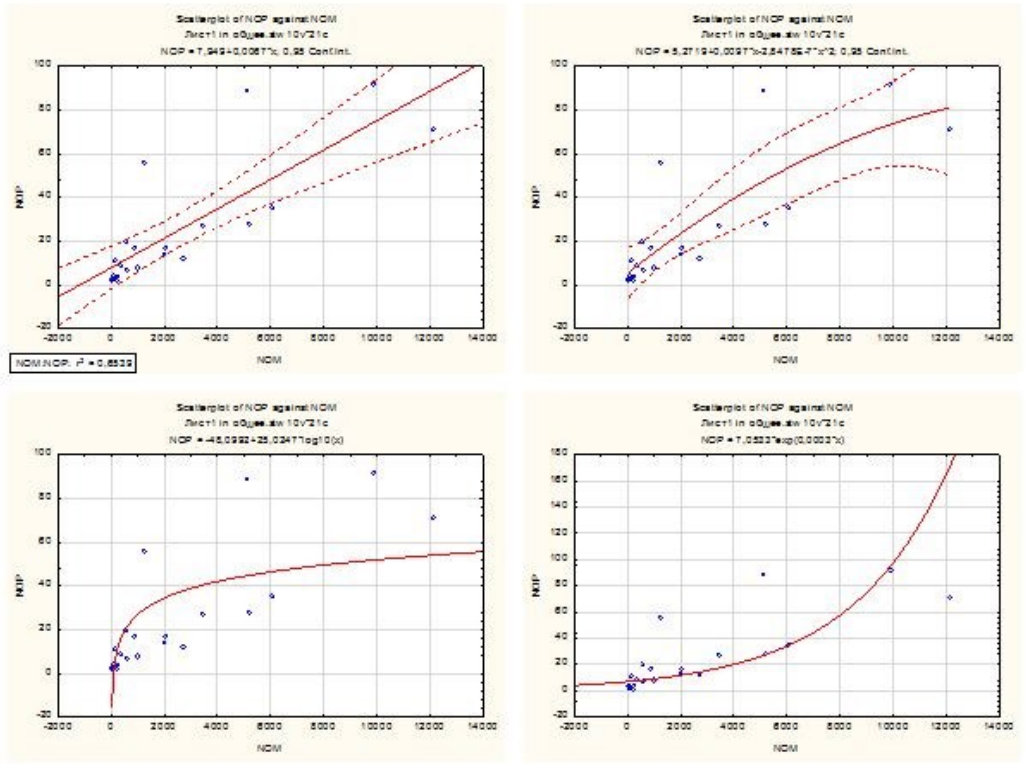


Рисунок 3.3 – Кореляційні залежності для пари NOP-NOM



Ці дані будуть в подальшому використовуватись для кореляційного та регресійного аналізу.

При проведенні кореляційного аналізу було за допомогою пакету Statistica обчислено коефіцієнти для пар метрика-експертна оцінка.

На основі отриманих даних, які подані в таблиці 4, можна зробити наступні висновки:

- значення коефіцієнтів кореляції для пар «метрика-експертна оцінка», що зображені червоним кольором вказують на залежність між метриками та експертною оцінкою;
- значення коефіцієнтів кореляції для пар «метрика-експертна оцінка», що зображені чорним кольором не залежні між собою(мала залежність);
- коефіцієнти кореляції не є дуже великим, що дає змогу сказати, що залежність між метриками та експертними оцінками, які досліджуються в даній курсовій роботі, не є значною.

Дані, що були отримані при проведені кореляційного аналізу, будуть використані з метою проведення регресійного аналізу.

Перелік пар метрик із значимими тісними залежностями: LOC-NOM; LOC-NOP; NOM-NOP; LOC-Зрозумілість.

Для проведення регресійного аналізу був використаний пакет Statistica. За його допомогою було автоматично побудовано лінії регресії для пар метрика-експертна оцінка, визначено функції регресії, обчислено відхилення і знайдено найбільш пов’язані пари.

На основі отриманих даних можна зробити наступні висновки:

- у пари метрик LOC-NOM присутня лінійна регресія;
- у пари метрик LOC-NOP присутня скоріш за все лінійна регресія;
- у пари метрик NOM-NOP присутня експоненціальна регресія;
- з побудованих графіків можна зробити висновок, що переважна більшість пар «метрика-експертна» дуже слабо пов’язані.

3.5 Статичний аналіз коду

Статичний аналіз коду (англ. Static code analysis) - аналіз програмного забезпечення, вироблений (на відміну від динамічного аналізу) без реального виконання досліджуваних програм. У більшості випадків аналіз проводиться над якою-небудь версією вихідного коду, хоча іноді аналізу піддається якій-небудь вид об'єктного коду, наприклад Р-код або код на MSIL. Термін зазвичай застосовують до аналізу, виробленому спеціальним ПО, тоді як річний аналіз називають "program understanding", "program comprehension" (розумінням або осягненням програми).

Залежно від використовуваного інструменту глибина аналізу може варіюватися від визначення поведінки окремих операторів до аналізу, що включає весь наявний вихідний код. Способи використання отриманої в ході аналізу інформації також різні - від виявлення місць, можливо з помилками (утиліти типу lint), до формальних методів, що дозволяють математично довести будь-які властивості програми (наприклад, відповідність поведінки специфікації).

Деякі люди вважають програмні метрики і зворотне проектування формами статичного аналізу. Отримання метрик і статичний аналіз часто поєднуються, особливо при створенні вбудованих систем (software quality objectives). [1]

Останнім часом статичний аналіз все більше використовується у верифікації властивостей ПЗ, використовуваного в комп'ютерних системах високої надійності, особливо критичних для життя (Safety-critical (англ.) Рос.). Також застосовується для пошуку коду, потенційно містить уразливості (іноді це застосування називається Static Application Security Testing, SAST). [2]

Статичний аналіз постійно застосовується в наступних областях:

ПЗ для медичних пристроїв. [3]

ПЗ для ядерних станцій і систем захисту реактора (Reactor Protection

Systems). [4]

ПЗ для авіації (в комбінації з динамічним аналізом) [5]

За даними VDC на 2012 рік, приблизно 28% розробників вбудованого ПЗ використовують засоби статичного аналізу, і 39% збираються почати їх використання протягом 2 років. [6]

Принципи статичного аналізу

Більшість компіляторів (наприклад, GNU C Compiler) виводять на екран «попередження» (англ. Warnings) - повідомлення про те, що код, будучи синтаксично правильним, швидше за все, містить помилку. наприклад:

```
int x;
int y = x + 2; // Мінлива x НЕ инициализирована!
```

Це найпростіший статичний аналіз. У компілятора є багато інших важливих характеристик - насамперед швидкість роботи і якість машинного коду, тому компілятори перевіряють код лише на очевидні помилки. Статичні аналізатори призначені для більш детального дослідження коду.

Типи помилок, що виявляються статичними аналізаторами]

• Невизначена поведінка - неініціалізовані змінні, звернення до NULL-вказівниками. Про найпростіших випадках сигналізують і компілятори.

• Порушення алгоритму користування бібліотекою. Наприклад, для кожного fopen потрібен fclose. І якщо файлова змінна втрачається раніше, ніж файл закривається, аналізатор може повідомити про помилку.

• Типові сценарії, що призводять до Недокументовані поведінки. Стандартна бібліотека мови Сі відома великою кількістю невдалих технічних рішень. Деякі функції, наприклад, gets, в принципі небезпечні. sprintf і strcpy безпечні лише при певних умовах.

• Переповнення буфера - коли комп'ютерна програма записує дані за межами виділеного в пам'яті буфера.

```
void doSomething (const char * x)
{
    char s [40];
```

sprintf (s, "[% s]", x); // Sprintf в локальний буфер, можливо переповнювання

....

}

• Типові сценарії, що заважають багатоплатформеності.

Object * p = getObject ();

int pNum = reinterpret_cast <int> (p); // На x86-32 вірно, на x64 частина покажчика буде втрачена; потрібен size_t

• Помилки в повторюваному коді. Багато програм виконують кілька разів одне і те ж з різними аргументами. Зазвичай повторювані фрагменти не пишуть з нуля, а розмножують і виправляють.

dest.x = src.x + dx;

dest.y = src.y + dx; // Помилка, треба dy!

• Помилки форматних рядків - у функціях зразок printf можуть бути помилки з невідповідністю рядка формату реальному типом параметрів. [7]

std :: wstring s;

printf ("s is% s", s);

• Незмінний параметр, переданий у функцію - ознака змінених вимог до програми. Колись параметр був задіяний, але зараз він вже не потрібен. У такому випадку програміст може взагалі позбутися цього параметра - і від пов'язаної з ним логіки.

```
void doSomething(int n, bool flag)   // flag всегда равен true
{
   if (flag)
   {
       // какая-то логика
   } else
   {
       // код есть, но не задействован
   }
}

doSomething(n, true);
...
doSomething(10, true);
...
doSomething(x.size(), true);
```

• Інші помилки - багато функцій зі стандартних бібліотек не мають побічного ефекту, і вик***лик їх як процедур не має сенсу***[7].

```
 std::string s;
...
s.empty();      // код ничего не делает; вероятно, вы хотели s.clear()?
```

3.6 Інструменти статичного аналізу

C/C++:
BLAST
Coverity
PC-Lint
lint и lock_lint, входящие в состав Sun Studio
Cppcheck (англ.)русск. (Cppcheck on sf)
Parasoft C/C++Test
SourceAnalyzer (также Fortran и x86 asm)
PVS-Studio (англ.)русск.
Java:
FindBugs (FindBugs on sf)
Parasoft JTest
.NET:
ReSharper
Python:
Pychecker
Pylint
Pyflakes

інші:

• T-SQL Analyzer - інструмент, який може переглядати програмні модулі в базах даних під керуванням Microsoft SQL Server 2005 або 2008 і виявляти потенційні проблеми, пов'язані з низькою якістю коду.

• АК-ВС (Пошук НДВ, [2], виявлення небезпечних шаблонів по CWE (англ.) Рос. [8])

4 ЕМПІРИЧНІ МЕТОДИ ОЦІНКИ НАДІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для кількісної оцінки показників надійності програмного забезпечення використовують різноманітні моделі надійності, під якими розуміються математичні моделі, побудовані для оцінки залежності надійності від заздалегідь відомих або визначених у ході виконання завдання параметрів. Ці моделі можна розділити на дві основні групи: емпіричні та аналітичні.

Емпіричні моделі засновані на аналізі накопиченої інформації про функціонування раніше розроблених програм.

Найбільш проста емпірична модель пов'язує число помилок у програмному забезпеченні з його обсягом. Досвідчені дані свідчать, що до початку системного тестування в програмному забезпеченні на кожні 1000 операторів припадає приблизно 10 помилок. Рівень надійності програмного забезпечення вважається прийнятним для початку експлуатації, якщо того ж об'єму операторів буде відповідати одна помилка.

Аналітичні моделі поділяються на статичні і динамічні. Серед динамічних, також можна виділити безперервні і дискретні.

При використанні безперервної динамічної моделі передбачається, що функціонування програмного забезпечення описується набором послідовних станів, перехід між якими відбувається у випадку виникнення відмови, за яким також слід відновлення. До безперервної динамічної моделі відносяться модель Джелінскі - Моранді і модель перехідних ймовірностей Маркова.

У дискретних моделях передбачається, що спочатку проводиться тестування програмного забезпечення (можливо, у кілька етапів). У разі появи відмов шукаються і виправляються всі помилки, через які сталися відмови. Після цього починається період експлуатації програмного забезпечення. Прикладами дискретних моделей можуть послужити модель Шумана і модель Мусса.

Статичні моделі відрізняються від динамічних насамперед тим, що в них не враховується час появи помилок. До статичні моделі відносяться модель Міллса, модель Нельсона і модель Коркорен.

Так як між надійністю і складністю програмного забезпечення існує тісний зв'язок, то проблем надійності стосується ще одна група моделей - моделі, призначені для оцінки складності програмного забезпечення. Ці моделі оцінюють безліч характеристик програмного забезпечення, таких як довжина програми, інформаційний зміст, число підсистем, число операторів, складність інтерфейсу і т.п. Всі існуючі моделі складності та метрики показників визначають тільки окремі, приватні характеристики складних програм. Загальним поняттям для всіх видів складних програм є їх структура.

При аналізі структурної складності програмного забезпечення з метою визначення його надійності необхідно проводити Багатокрокове процедуру зниження його складності. Поширення складних програмних систем і комплексів вимагає нових підходів до оцінки їх надійності. Складність вирішення цього завдання обумовлена відсутністю універсальних методів і моделей.

Для зручності аналізу показників надійності складних програмних комплексів доцільно представити їх у вигляді сукупності менш складних складових, програмних модулів (ПМ). Такими модулями можуть бути програмні комплекси, окремі програми, блоки або оператори. Кількість модулів у програмному комплексі може бути занадто великим для обробки, тому частіше програмні модулі групують за типами. Кожен тип містить програмні модулі, близькі за властивостями, у тому числі по надійності. За заданою структурі програмного комплексу, що складається з деякої сукупності програмних модулів, що мають відомі показники надійності, існує можливість знайти показник надійності програмного комплексу. Для цього використовуються так звані графові моделі програми (ГМП).

В якості графовой моделі програми розглянемо орієнтований граф G (V, Г), де V = {vi} - безліч вершин, Г = {gij} - безліч дуг. Граф системи G (V, Г) визначається структурою програмної системи. Безліч V вершин графа складає програмні модулі (типи модулів), а множина дуг Г відображає зв'язок між модулями, тобто, якщо з i-го модуля є перехід в j-ий модуль, то в графі G є дуга gij, ведуча з i- ой вершини в j-ую. Введемо модельні обмеження. Припустимо наявність в графовой моделі програми однієї початкової вершини v0 (вхід) і однієї кінцевої vk (вихід). Припустимо також, що з кожної вершини виходить не більше двох дуг, число входять у вершину дуг не обмежується. Будемо вважати, що графовая модель програми не містить циклів, а відображається нею програма належить до категорії несамозмінних.

При моделюванні обчислювального процесу на графової моделі програми та дослідженні властивостей програмного забезпечення передбачається повідомлення кожному елементу моделі деякого ваги. Припустимо, що кожна

вершина vi характеризується адитивним елементарним показником di, пов'язаних з досліджуваним властивістю програми. Введені показники утворюють на графі безліч D = {di}. Модель G (V, Г, D) може використовуватися для статистичного дослідження різних маршрутів графової моделі програми. Вибір шляху прогону на графі обумовлюється сукупністю реалізацій передач управління у вершинах, які пов'язані з випадковим процесом надходження на вхід програми різних векторів вхідних даних, що призводить до випадкового характеру вибору маршрутів в графі. Таким чином, досліджуване програмне забезпечення можна представити складною системою з випадковою структурою, динаміку функціонування якої доцільно описати статистично за допомогою ймовірностей переходу від і-ої до j-ой вершині графової моделі програми.

Для завершальній стадії життєвого циклу програмного комплексу слід використовувати модель визначення надійності з системно-незалежною аргументом (кількість прогонів ПО), наприклад, модель Нельсона.

Однак практичне використання цієї моделі викликає труднощі, особливо для відносно великих програмних комплексів масового застосування, оскільки пов'язує оцінку надійності програмного забезпечення з кількістю можливих програмних маршрутів реалізації обчислень і не розглядає характеристики цих маршрутів. Для усунення цих недоліків моделі Нельсона - єдиною, що визначає надійність програмного забезпечення в період експлуатації, - використовують структурні графові моделі досліджуваного програмного забезпечення.

Є й інші моделі, які в тому чи іншому вигляді дають оцінки надійності. Моделі ці часто використовують будь-яку додаткову інформацію про процес розробки, наприклад дані про покриття коду тестами або деяку інформацію про організацію процесу розробки. Як наслідок, можливість застосування таких моделей часто обмежена, через що вони знаходять мало застосування на практиці.

5 ОБРОБКА ТА УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ ЕКСПЕРИМЕНТІВ

Методологічна функція процедури обробки результатів наукової праці полягає у встановленні ступеня їхньої адекватності досліджуваному явищу, точності й надійності, на підставі яких у вербальному або формалізованому вигляді узагальнюється здобуте теоретичне чи емпіричне знання про динамічні або статистичні закономірності, закони, принципи, що визначають взаємодію причини й наслідку в оточуючому людину світі.

Під час логіко-змістової обробки та узагальнення теоретичних результатів наукового пошуку головної ролі набуває необхідність обґрунтування їхньої адекватності як придатності для вірного формування й відтворення у мисленні людини уявлень про зв'язки та відношення, відповідні до тих, що існують в об'єктивному світі.

Результати теоретичних досліджень чи розробки будь-яких наукових теоретичних конструктів незалежно від форми викладу (вербальний або формалізований) слід перевіряти на адекватність за критерієм відповідності природі або сутності описуваних ними явищ.

При цьому методологічним інструментарієм перевірки результатів розробки теоретичного знання на адекватність слугують логіка діалектичного мислення, формальна та математична логіка й усі існуючі загальнонаукові та пов'язані з ними конкретно наукові методи, а сама перевірка носить переважно опосередкований характер.

Експериментальні дані узагальнюють у такій послідовності: знаходять функціональні зв'язки між" досліджуваними характеристиками або параметрами, встановлюють основні закономірності взаємодії розглядуваних параметрів, розкривають природу та сутність причинно-наслідкового зв'язку, Що має місце між аналізованими параметрами, виражають наявний зв'язок у математичній формі, роблять відповідні теоретичні узагальнення.

В основу переважної більшості використовуваних наукою методів

узагальнення результатів експериментальних досліджень покладено ідеї ізоморфізму (природна або сутнісна особливість об'єкта, явища чи процесу, завдяки якій характер закономірностей взаємодії причин і наслідків у них є тотожний), та інваріантності (властивість суттєвих характеристик, величин і показників об'єкта чи процесу не змінюватися при їхніх певних перетворюваннях).

На зазначених ідеях базуються широко відомі унормовування експериментальних даних, узагальнення змінних та інші методи обезрозмірювання отримуваних показників. Сутність вказаних методів полягає в переході від абсолютних значень вимірюваних параметрів до відносних одиниць їхніх оцінок. Необхідність розробки й застосування цих методів пов'язана саме з тим, що науку цікавлять здебільшого відносні зміни станів певного класу чи груп досліджуваних об'єктів або явищ. Саме на виявлення змін і спрямовано переважно методи: експертних оцінок, ранжирування, безпосередніх оцінок, багатомірного шкалування, узагальнених змінних тощо.

Найкращим способом перевірки адекватності експериментально встановлених закономірностей досліджуваному явищу слугує їхня апробація практикою, оскільки, як відомо, будь-яким емпіричним закономірностям притаманна відтворюваність.

Щодо методів аналізу та математичної обробки результатів експериментальних досліджень, то їх прийнято поділяти на графічні й аналітичні. Для пошуку функціональних зв'язків між експериментальними даними використовують таблиці та графіки. На відміну від таблиць, які допомагають з'ясовувати функціональні зв'язки, графіки дають можливість спостерігати зміни функцій як на площині, так і в просторі. Графік більш зрозуміло, ніж таблиця відбиває розвиток явища у функціональному зв'язку та закон цього розвитку. Проте таблиці більш точно відображають результати вимірювань (у таблиці можна розташовувати до 10 і навіть більше залежних змінних, у той час як на графіку добре сприймаються лише криві трьох-чотирьох функцій).

Тому під час проведення дослідження слід користуватися і таблицями, і побудованими за ними графіками, а в текст звіту або дисертаційної праці при можливості доцільно вміщувати переважно графіки та переносити значну частину таблиць у матеріали додатків.

До найбільш типових методів аналітичної (математичної) обробки експериментальних даних відносяться певні різновиди їхнього математичного й графічного згладжування та кореляції, гармонійний аналіз, знаходження оптимумів, дисперсійний аналіз тощо. Оцінку точності вимірювань, виконуваних під час експериментальних досліджень, та надійності отриманих при цьому результатів здійснюють на основі теорії похибок із застосуванням методів математичної статистики.

Зважаючи на викладене, розглянемо дещо детальніше ряд найбільш частотних за вживанням у різних сферах наукового пошуку методів обробки та узагальнення результатів експериментального дослідження.

5.1 Методи експертних оцінок

Методи експертних оцінок використовують, як правило, для виявлення пріоритетних напрямів досліджень, планування багатофакторних експериментальних досліджень для оцінки значущості факторів, а також у багатьох інших випадках наукової практики, коли виникає необхідність у прийнятті рішень в умовах альтернативного вибору. Найбільшого поширення У дослідженнях набули методи ранжирування й безпосередньої оцінки.

Під ранжируванням розуміють метод або процедуру упорядкування об'єктів, предметів, властивостей, ознак, факторів тощо, спрямовану на встановлення їхньої відносної значущості.

5.1.1 *Методи ранжирування.*

Необхідність у використанні методу ранжирування виникає у випадку, коли через відсутність потрібного теоретичного знання або недостатньої вивченості властивостей, зв'язків чи відношень об'єкта дослідник змушений

спиратися на інтуїтивне, імовірнісне бачення проблеми кваліфікованими експертами.

Процедура реалізації методу ранжирування складається із послідовності таких дій: 1) формулювання завдання експертної оцінки, 2) формування низки альтернативних об'єктів (або ряду альтернативних факторів), 3) ранжування альтернатив експертами, 4) обробка дослідником результатів ранжирування.

Завдання експертної оцінки має бути однозначним і включати основну ознаку або критерій оцінки (наприклад: "Встановити значущість наведених факторів за їхнім впливом на ефективність процесу практичного навчання учнів середніх шкіл", "З'ясувати значущість психічних якостей людини, що сприяють прискоренню її виходу із стресової ситуації", "Здійснити ранжирування промислових об'єктів за ступенем їхньої пожежної небезпеки").

Формування низки альтернатив полягає в тому, що дослідник записує їх у головці відповідної експертної таблиці (див., наприклад, Табл. 5.1) у довільному порядку. У бокову частину таблиці заносять прізвища експертів та показники результатів обробки проведеного ранжирування. Зауважимо тут, що доцільним вважається ранжирування не менше 7 й не більше 20 об'єктів.

Ранжирування альтернатів експертами передбачає визначення місця або рангу кожного об'єкта чи фактора у запропонованому їхньому ряді. Для цього експерт приписує кожному фактору свій ранг, позначаючи його одним із чисел натурального ряду (1, 2, 3, 4, 5 і т.д.). При цьому експерт найбільш значущу, на його думку, альтернативу повинен помічати цифрою "1", яка засвідчує пріоритетну значущість поміченого нею об'єкта чи фактору. Таким чином, найбільша цифра натурального ряду означатиме, що помічений нею об'єкт чи фактор є найменш значущим в альтернативному ряді.

Таблиця 5.1.- Приклад матриці для ранжирування факторів, що впливають на ефективність практичного навчання учнів середніх шкіл

Експерти	Фактори			
Тендерні особливості учня (Фі)	Методика навчання ($Ф_2$)	Бажання учня оволодіти трудовими навичками (Фз)	Темперамент учня ($Ф_4$)	
Ei(Архипов Н.П.)	4	2	1	
E_2 (Буряк СМ.)	3	1	2	
Е3 (Волошко П.Д.)	4	3	1	
Сума експерт-них оцінок	11	6	4	
Результуючий ранг	IV	II	I	II

5.1.2 *Обробка результатів ранжування*

Обробка дослідником результатів ранжирування проводиться таким чином. Дослідник підраховує суму цифр, проставлених експертами в таблиці по кожному окремо взятому фактору й заносить її у відповідну клітинку передостаннього рядка таблиці експертних оцінок (див. у нашому прикладі рядок "Сума експертних оцінок"). Після цього він визначає результуючі ранги за таким правилом: найвищий (I) ранг присвоюється об'єкту або фактору, що отримав найменшу суму експертних оцінок, а найнижчий ранг - об’єкту, що

отримав їхню найбільшу суму (у нашому прикладі найнижчий (IV) результуючий ранг присвоєно тендерному фактору).

Таким чином, наведені в прикладі результати ранжирування показують, що за ступенем значущості, фактори, яки впливають на ефективність практичного навчання учнів середніх шкіл, слід враховувати в такому порядку: I - бажання учня оволодіти трудовими навичками, II - методика навчання, III - темперамент учня, IV - тендерні особливості учня.

Проте метод ранжирування дозволяє визначити лише місце, яке за ступенем значущості впливу на наслідок займає кожний із факторів альтернативного ряду, тобто - віддзеркалити лише якісний бік їхнього зв'язку.

Метод безпосередньої оцінки. Цей метод полягає в тому, що діапазон зміни будь-якої якісної змінної поділяють на декілька оцінних інтервалів, Кожному із зазначених інтервалів присвоюють певну оцінку (або бал), наприклад від 0 до 10 (від 0 до 1 або від 0 до 100).

Під методом безпосередньої оцінки розуміють метод або процедуру упорядкування об'єктів, предметів, властивостей, ознак, факторів тощо за кількісними показниками значущості кожного об'єкта, віднесеної до суми значимостей об'єктів всього альтернативного ряду.

Завдання експерта полягає у віднесенні кожного із об'єктів (факторів чи альтернатив) до певного оціночного інтервалу або у присвоюванні кожному об'єкту певного оціночного балу. Така безпосередня оцінка ґрунтується на імовірнісних судженнях експерта про ступінь наявності в об'єкта тих чи інших властивостей або на припущеннях про їхню значущість.

Використання основаного на процедурі ранжирування методу безпосередньої експертної оцінки, по-перше, надає дослідникові можливість встановлювати якісну характеристику взаємозв'язку об'єктів у межах певного альтернативного ряду, яка відображає їхню відносну значущість. По-друге, а це особливо важливо, дозволяє отримати кількісні ознаки згаданого взаємозв'язку через визначення відносної або відсоткової ваги кожного

об'єкта цього ряду.

Найбільш простим випадком ранжирування за методом безпосередньої оцінки є випадок, у якому на запитання анкети експерту пропонують відповідати **"так"** або "**ні**". При цьому відповіді "так" присвоюється бал 1, а відповіді "ні" - бал 0. Для безпосередньої оцінки може бути задано й триступеневу шкалу: **"дуже важливо" -** бал **2, "важливо" -** бал **1, "не має значення"** - бал **0.**

Оцінка ступеня інтенсивності будь-якої властивості або ознаки може проводиться, наприклад, за такою шкалою балів: "надзвичайно низький" - 0, "низький" - 1, "середньопонижений" - 2, "середній" - 3, "середньо підвищений" - 4, "високий" - 5, "надзвичайно високий" - 6.

При цьому важливо лише те, щоб кожному балу відповідало цілком визначене поняття, здатне виражати ступінь інтенсивності властивості, якості або якоїсь іншої ознаки.

Застосування різних кількісних шкал і прийомів для систематизації інформації, отриманої від експертів, проводиться відповідно до таких правил: 1) сума оцінок окремих об'єктів альтернативного ряду повинна бути рівною одиниці, 2) оцінка будь-якого об'єкта альтернативного (взаємовиключного) ряду має виражатися числом не менше нуля й не більше одиниці, 3) при об'єднанні двох або більшої кількості об'єктів в одне ціле, оцінка, приписана цьому цілому, повинна дорівнювати сумі оцінок вихідних об'єктів.

Пов'язана з теорією ймовірностей необхідність дотримання наведених нами правил побудови кількісних шкал дозволяє формувати певну несуперечливу систему оцінок навіть в умовах існування суб'єктивізму бачення проблеми експертами.

5.1.3 *Нормування отриманих оцінок*

Проте у практиці наукових досліджень часто виникає потреба у використанні суми оцінок, рівній не одиниці, а якомусь іншому фіксованому числу. Вона може зумовлюватися і певною зручністю визначення оцінок для

експерта, і спрощенням розрахунків для дослідника. У таких випадках, тобто при застосуванні експертами* довільних шкал (див. Табл. 5.2), дослідник з метою дотримання наведених вище правил має проводити нормування оцінок.

Таблиця 5.2. Приклад матриці реєстрації та обробки результатів безпосередніх експертних оцінок при застосуванні довільних шкал

Експерти	Фактори				
$Ф_1$	ф 2	ф 3	ф 4		
Е,	15	25	20	70	45
$Е_2$	4	3	8	9	5
$Е_3$	45	50	60	120	95

Нормовані оцінки

Е,	0,086	0,143	0,114	0,400	0,257
$Е_2$	0,138	0,103	0,276	0,311	0,172
$Е_3$	0,122	0,135	0,162	0,323	0,258
Усереднена оцінка	0,115	0,127	0,184	0,345	0,229
Результуючий ранг	V	IV	III	I	II

Для цього всі оцінки, дані одним експертом (див., наприклад, рядок оцінок першого експерта - Е*i* у таблиці 5.2), підсумовуються, а потім кожну з них ділять на отриману суму. Результат кожного зазначеного поділу заносять Іу клітинку відповідного фактору, у верхній рядок другої таблиці матриці (див. рядок - Еі таблиці "Нормовані оцінки"). Нормовані (тобто виражені у долях від одиниці) у такий спосіб оцінки усереднюються окремо по кожному

фактору. За результатами усереднення визначають результуючі ранги кожного фактору, надаючи перший (I) - найвищий ранг - фактору, що отримав найбільшу усереднену оцінку. Зрозуміло, що найнижчий ранг (V) отримає при цьому фактор із найнижчою усередненою оцінкою.

Дуже зручним вважається також спосіб встановлення залежностей між факторами, при якому експерту пропонується присвоювати найбільш важливому на його думку фактору оцінку рівну наперед заданому числу, а оцінки наступних за важливістю факторів визначати як долю від більш важливого. Основна перевага такого способу полягає в тому, що експерту немає потреби кожного разу зіставляти усі оцінки. За цим способом достатньо лише враховувати значення першої й попередньої за важливою оцінок.

Для виробки навичок методологічної рефлексії доцільно розглянути на прикладі Табл. 5.3 порядок реалізації процедур, що входять до складу метода безпосередньої експертної оцінки, у якому вживане присвоєння найбільш важливому фактору наперед заданої оцінки, що дорівнює числу 100.

Таблиця 5.3 - Приклад матриці реєстрації та обробки результатів безпосередніх експертних оцінок для випадку присвоєння найбільш важливому фактору попередньо заданої оцінки

Експерти	Фактори						
$Ф_1$	$Ф_2$	Фз	Ф4	Ф5	Фб	Ф7	
Е,	5	100	75	30	85	6	3
$Е_2$	8	65	100	70	90	7	2
$Е_3$	6	60	80	90	100	10	5

Нормовані оцінки

Е,	0,016	0,329	0,247	0,099	0,280	0,020	0,009
$Е_2$	0,023	0,190	0,292	0,205	0,263	0,021	0,006

E_3	0,017	0,171	0,228	0,257	0,285	0,028	0,014
Усереднена оцінка (%)	1,8	23,0	25,6	18,7	27,7	2,3	1.0
Результуючий ранг	VI	III	II	IV	I	V	VII

У наведеному прикладі оцінку альтернативного ряду факторів здійснено групою, до складу якої входять три експерти ($E_ь$ E_2, E_3). При цьому, що є цілком природним, кожний експерт має свою шкалу преференцій.

Завдання полягає у виявленні ступеня очікуваного впливу кожного з факторів (<X>і, $Ф_2$, Фз-.. Ф7) ^{на} ефективність реалізації певного процесу. Очевидно, що експертам запропоновано присвоювати найбільш важливому фактору оцінку, яка дорівнює числу 100, тобто визначено шкалу оцінок від 0 до 100 балів. Результати експертних оцінок занесені в матрицю, у верхню її частину у вигляді таблиці "експерт - фактор".

На підставі даних експертних оцінок дослідником здійснюються обробка й узагальнення результатів безпосередніх оцінок. Для цього, насамперед, обраховуються суми оцінок кожного експерта:

$\sum_{1}^{n} G_n^k = G_1 + G_2 = G_3 + \dots + G_m$, де n – порядковий номер оцінюваного фактора, k – порядковий номер експерта. У нашому прикладі за цією формулою буде отримано такі результати:

$$\sum_{1}^{7} G_7^1 = 5 + 100 + 75 + 30 + 85 + 6 + 3 = 304,$$

$$\sum_{1}^{7} G_7^2 = 8 + 65 + 100 + 70 + 90 + 7 + 2 = 342,$$

$$\sum_{1}^{7} G_7^3 = 6 + 60 + 80 + 90 + 100 + 10 + 5 = 351.$$

Далі розраховуються відповідні значущості кожного фактора (G_{kn}):

$G_{kn} = \frac{G_n}{\sum_{1}^{n} G_n^k}$, де G_n – оцінка окремого фактора, занесена в таблицю "експерт-фактор", n – порядковий номер фактора, k – порядковий номер експерта.

$$G_{11} = \frac{5}{304} = 0{,}016; \quad G_{12} = \frac{8}{342} = 0{,}023; \quad G_{13} = \frac{6}{351} = 0{,}017;$$
$$G_{21} = \frac{100}{304} = 0{,}329; \quad G_{22} = \frac{65}{342} = 0{,}190; \quad G_{23} = \frac{60}{351} = 0{,}171;$$
$$G_{31} = \frac{75}{304} = 0{,}247; \quad G_{32} = \frac{100}{342} = 0{,}292; \quad G_{33} = \frac{80}{351} = 0{,}228;$$
$$G_{41} = \frac{30}{304} = 0{,}099; \quad G_{42} = \frac{70}{342} = 0{,}205; \quad G_{43} = \frac{90}{351} = 0{,}257;$$
$$G_{51} = \frac{85}{304} = 0{,}280; \quad G_{52} = \frac{90}{342} = 0{,}263; \quad G_{53} = \frac{100}{351} = 0{,}285;$$
$$G_{61} = \frac{6}{304} = 0{,}020; \quad G_{62} = \frac{7}{342} = 0{,}021; \quad G_{63} = \frac{10}{351} = 0{,}028;$$
$$G_{71} = \frac{3}{304} = 0{,}009; \quad G_{72} = \frac{2}{342} = 0{,}006; \quad G_{73} = \frac{5}{351} = 0{,}014.$$

Обраховані таким чином або унормовані дані заносяться в таблицю "Нормовані оцінки" розглядуваної нами матриці (табл. 5.3). Після цього у відсотках від загальної значущості факторів альтернативного ряду обчислюються усереднені для трьох експертів оцінки окремих факторів:

$$G_{cp}^{n} = \frac{\sum_{1}^{k} G_{kn}^{k}}{l} \times 100 = \frac{G_{kn}^{1} + G_{kn}^{2} + G_{kn}^{3}}{l} \times 100(\%), \quad \text{де} \quad k$$

порядковий номер експерта, / - кількість експертів, що орали участь в оцінюванні значущості факторів. Виконання розрахунків відповідно до цієї формули дає такі результати:

$$G_{cp}^{1} = \frac{0,016 + 0,023 + 0,017}{3} \times 100 = \mathbf{1,8\%};$$

$$G_{cp}^{2} = \frac{0,329 + 0,190 + 0,171}{3} \times 100 = \mathbf{23,0\%};$$

$$G_{cp}^{3} = \frac{0,247 + 0,292 + 0,228}{3} \times 100 = \mathbf{25,6\%}$$

$$G_{cp}^{4} = \frac{0,090 + 0,205 + 0,257}{3} \times 100 = \mathbf{18,7\%};$$

$$G_{cp}^{5} = \frac{0,280 + 0,263 + 0,285}{3} \times 100 = \mathbf{27,7\%};$$

$$G_{cp}^{6} = \frac{0,020 + 0,021 + 0,028}{3} \times 100 = \mathbf{2,3\%};$$

$$G_{cp}^{7} = \frac{0,009 + 0,006 + 0,014}{3} \times 100 = \mathbf{1,0\%}.$$

Визначені показники усереднених експертних оцінок значущості кожного фактора заносяться у відповідний рядок матриці. Фактору, що отримав найбільшу усереднену оцінку, присвоюється найвищий (I) ранг, наступному за ним по величині усередненої оцінки фактору присвоюється II ранг і т.д. Показники рангів значущості факторів, які позначаються римськими цифрами, також закосяться у відповідний рядок матриці -"Результуючий ранг".

Оброблені таким чином результати безпосередньої експертної оцінки аналізуються дослідником й узагальнюються у вербальній формі. Так, із табл. З видно, що з семи оцінюваних факторів лише чотири мають суттєвий вплив на ефективність реалізації розглядуваного експертами процесу. При цьому за ступенем зазначеного впливу вони розташовуються в такій послідовності: $Ф_5$, Фз, Фг і Ф4- Сумарна значущість цих чотирьох факторів складає майже 95% від загальної значущості всіх факторів альтернативного ряду. Звідси стає очевидним також, що під час проведення багатофакторного експериментального дослідження є сенс враховувати лише вплив чотирьох факторів ($Ф_2$, Фз, Ф4, Ф5) на ефективність процесу.

Значного поширення в науковій практиці набув також **спосіб безпосередньої експертної оцінки з використанням попередньо заданої шкали рівнів** варіювання певних ознак (властивостей, характеристик, показників, параметрів) ряду об'єктів або характеристик змін стану складних процесів чи явищ, що досліджуються експериментальним шляхом.

Згідно з цим способом експертам необхідно у відповідності із запропонованою дослідником шкалою здійснити безпосередні оцінки рівнів прояву конкретної ознаки в кожного з розглядуваних об'єктів або ступенів зміни ознак того ж самого об'єкта.

Приклад матриці реалізації методу безпосередньої експертної оцінки з використанням попередньо заданої шкали для встановлення ступеня прояву досліджуваної ознаки в ряді подібних об'єктів відображено в табл. 6. Матрицю такого типу може бути використано в межах проведення педагогічного експерименту.

Наприклад, дослідник за допомогою методу ранжирування виявив, що найбільш значущим фактором, який впливає на ефективність практичного навчання учнів середніх шкіл, є бажання учня оволодіти трудовими навичками (див. табл. 5.1, фактор Фі). Цілком природно уявити при цьому, що він включає до завдань експериментального педагогічного дослідження питання встановлення ступеня прояву зазначеного бажання у кожного з учнів 7-х класів середньої школи.

Уявимо, що для вирішення цього питання дослідник застосовує метод безпосередньої оцінки, завдання якого сформульовано так: "Встановити ступені намагання учнів 7-х класів середніх шкіл оволодіти трудовими навичками".

Таблиця 5.4. - Приклад фрагменту матриці результатів реєстрації безпосередніх експертних оцінок за попередньо заданою шкалою

Об'єкт або характеристика зміни стану об'єкта (прізвище учня)	Експерти	Ступінь прояву ознаки (ступенів намагання учнів 7-х класів середніх шкіл оволодіти трудовими навичками)						
		Надзвичайно високий	Високий	Середньо-підвищений	Середній	Середньо-понижений	Низький	Надзвичайно низький
		I	II	III	IV	V	VI	VII
1. Бурмака П.	E_1		●					
	E_2			●				
	E_3			●				
усереднена оцінка				●				
2. Бриль М.	E_1				●			
	E_2			●				
	E_3				●			
усереднена оцінка					●			
3. Вовк Н.	E_1					●		
	E_2					●		
	E_3					●		
усереднена оцінка						●		
4. Гетьман О.	E_1			●				
	E_2			●				
	E_3		●					
усереднена оцінка				●				
5. Доренко Ф.	E_1				●			
	E_2					●		
	E_3	•			●			
усереднена оцінка					●			
6. Живило С.	E_1			●				
	E_2				●			
	E_3			●				
усереднена оцінка				●				

На цих підставах ним обирається семирівнева шкала інтенсивності або ступенів намагання оволодіння трудовими навичками, яка охоплює такі рівні: надзвичайно високий (I), високий (II), середньопідвищений (III), середній (IV), середньопонижений (V), низький (VI) та надзвичайно низький (VII). Експертам пропонується на основі безпосередніх прослуховувань відповідей учнів на теоретичних уроках практичного навчання та спостережень дій під час практичних занять оцінити ступінь намагання кожного з них оволодіти трудовими навичками.

Згідно з результатами експертних оцінок дослідником складається відповідна матриця, приклад фрагмента якої наведено в Табл. 5.4. Із таблиці видно, що за даними експертів ступінь прояву учнями 7-х класів намагань

оволодіти трудовими навичками варіює від високого до середньо-пониженого. При цьому відповідно до даних фрагмента матриці за усередненими оцінками експертів (виділено тонуванням) середньо-підвищений ступінь намагань є притаманним 50% учнів, середній ступінь - 33,3%, а середньопонижений - 16,7%.

Зазвичай у процесі проведення безпосередніх експертних оцінок за попередньо заданою шкалою виникає необхідність використання більш складних матриць. Розглянемо порядок побудови такої матриці на прикладі Табл. 5.5.

Теоретичним підґрунтям для складання цієї матриці слугувала робоча класифікація ознак актуалізації функціонально-семантичних типів і видів висловлювань-невдоволень, відображена на рис. 29. У відповідності із зазначеною класифікацією методом ранжирування факторів або на основі логічного обґрунтування, здійсненого під час аналізу стану розробленості проблеми наукової праці, дослідником було встановлено, що на особливості просодичного оформлення висловлювань-невдоволень здійснюють найсуттєвіший вплив такі фактори: соціокультурний рівень мовця, ситуація спілкування та почуття, які виражає мовець. При цьому було з'ясовано, ідо вплив зазначених факторів має випадковий характер, а, отже, його кількісну оцінку можливо отримати лише у певному імовірнісному вигляді.

Виходячи з цього, дослідник формулює завдання процедури безпосередньої оцінки: "Установити вплив соціокультурного рівня мовця на частоту актуалізації ним різновидів гучності у висловлюваннях-невдоволеннях, що передають різні почуття". Для здійснення безпосередніх оцінок результатів зазначеного впливу експертам пропонується відповідна шкала гучності: висока, підвищена, помірна, знижена, низька.

Таблиця 5.5.- Вплив соціокультурного рівня мовця на частоту актуалізації гучності у висловлюваннях-невдоволеннях:

а) формальна ситуація спілкування (%)

Соціокультурний рівень	Вираження почуття	Гучність				
		висока	підвищена	помірна	знижена	низька
високий	гнів	2.00	2.00	2.00	0.00	0.00
	обурення	1.00	1.00	5.00	1.00	0.00
	роздратування	1.00	3.00	8.00	1.00	1.00
	образа	0.00	0.00	7.00	0.00	0.00
	протест	0.00	0.00	5.00	0.00	0.00
	презирство	1.00	12.00	21.00	5.00	1.00
	незадоволення	1.00	1.00	9.00	0.00	0.00
	досада	1.00	1.00	0.00	1.00	0.00
	сум	0.00	0.00	3.00	2.00	0.00
середній	гнів	0.00	0.00	0.00	0.00	0.00
	обурення	0.00	0.00	4.71	0.00	0.00
	роздратування	0.00	2.35	3.53	0.00	0.00
	образа	0.00	0.00	4.71	0.00	0.00
	протест	0.00	12.94	25.88	0.00	0.00
	презирство	0.00	3.53	12.94	1.18	0.00
	незадоволення	0.00	1.18	3.53	0.00	0.00
	досада	0.00	0.00	2.35	0.00	0.00
	сум	0.00	0.00	1.18	4.71	0.00
низький	гнів	2.82	3.52	0.00	0.00	0.00
	обурення	2.82	3.52	0.00	0.00	0.00
	роздратування	0.70	1.41	4.23	0.00	0.00
	образа	1.41	5.63	12.68	0.70	0.00
	протест	4.93	7.75	11.27	2.11	0.00
	презирство	1.41	1.41	0.00	0.70	0.00
	незадоволення	0.00	0.70	3.52	0.00	0.00
	досада	0.00	0.70	2.82	1.41	0.70
	сум	0.00	1.41	2.82	0.00	0.00

б) неформальна ситуація спілкування (%)

Соціокультурний рівень	Вираження почуття	Гучність				
		висока	підвищена	помірна	знижена	низька
високий	гнів	5.92	0.66	0.00	0.00	0.00
	обурення	0.66	0.66	0.66	0.00	0.00
	роздратування	0.66	3.95	2.63	0.66	0.66
	образа	1.97	1.32	3.29	0.00	0.00
	протест	1.97	6.58	11.84	3.29	0.00
	презирство	0.66	3.95	9.87	0.00	0.00
	незадоволення	0.66	2.63	7.24	0.00	0.00
	досада	0.00	0.00	1.32	0.66	0.00
	сум	0.00	0.00	5.26	5.92	0.00
середній	гнів	1.02	3.06	4.08	1.02	0.00
	обурення	0.00	15.31	0.00	0.00	0.00
	роздратування	1.02	0.00	2.04	0.00	0.00
	образа	0.00	0.00	11.22	13.27	1.02
	протест	1.02	2.04	1.02	2.04	0.00
	презирство	0.00	3.06	0.00	0.00	0.00
	незадоволення	2.04	4.08	2.04	1.02	0.00
	досада	0.00	0.00	3.06	0.00	0.00
	сум	0.00	0.00	4.08	4.08	0.00
низький	гнів	1.71	3.42	4.27	0.00	0.00
	обурення	0.00	5.13	4.27	0.00	0.00
	роздратування	2.56	0.85	1.71	0.00	0.00
	образа	0.00	2.56	2.56	0.00	0.00
	протест	0.00	4.27	0.85	0.00	0.00
	презирство	5.13	10.26	16.24	0.85	0.00
	незадоволення	0.00	0.00	6.84	0.00	0.00
	досада	0.85	0.00	0.85	3.42	0.00
	сум	0.00	0.00	5.13	0.43	0.00

На цих підставах оформлюється таблиця, у якій відображають кількісні (частотні показники актуалізації гучності у двох ситуаціях спілкування) та якісні (рівні реалізації гучності: висока, підвищена тощо) показники, отримані шляхом обробки й узагальнення безпосередніх результатів експертних оцінок. Така таблиця або матриця будується, як це відмічалося вище, для розкриття впливу, який здійснює сукупність певних причин на наслідок та виявлення значущості впливу кожної з причин. Тому ознаки, характеризуючи рівні варіювання сукупності причин, наводять у боковині матриці (див. табл. 7), шкалу оцінок експертами змін гучності відображають у головці матриці. У сформовану таким чином таблицю заносять результати розрахунків частотних показників актуалізації гучності в різних (формальній та

неформальній) ситуаціях спілкування. Якісний бік впливу соціокультурного рівня мовця на специфіку актуалізації гучності у висловлюваннях-невдоволеннях відображають шляхом тонування клітинок матриці, у яких зазначено відсоткові показники частоти реалізації конкретного рівня гучності. Тонування здійснюють так, що більшому відсотковому показнику відповідає більша інтенсивність тону. Завдяки цьому більш зручним стає загальний опис, оскільки, наприклад, за інтенсивністю тонування таблиці 5.5 неважко дійти висновку, що в реалізаціях переважної більшості висловлювань-невдоволень домінує підвищена і помірна гучність. Для отримання того ж самого висновку у разі відсутності тонування необхідно було б здійснювати поступове зіставлення значної кількості цифрових показників таблиці.

Звернемо окрему увагу на те, що основні методологічні умови застосування методу безпосередньої експертної оцінки за наперед заданими шкалами полягають у необхідності виконання дослідником таких вимог щодо формування шкали оцінок.

1. Фактор, зміну станів якого віддзеркалює шкала оцінок, має бути за своєю сутністю ні чим іншим, як наслідком імовірнісної, недостатньо дослідженої на даний час взаємодії певної сукупності інших факторів, що відіграють роль причини зазначених змін.

2. Оцінки, призначені для відображення ступеня прояву оцінюваного фактору, слід розташовувати на шкалі у векторній послідовності, що відповідає природній послідовності його змін: від меншого ступеня до більшого або навпаки.

3. Шкала оцінок має бути рівномірною, тобто відсоткові показники кожної з її оцінювальних зон, віднесені до суми показників усіх зон шкали, повинні бути однаковими.

4. Шкали експертних оцінок ступеня або інтенсивності прояву досліджуваної ознаки можуть охоплювати лише непарну кількість ступенів зміни цієї ознаки.

5. Шкали експертних оцінок форми або будь-яких зовнішніх ознак

можуть мати і парну, і непарну кількість ступенів їхніх оцінок.

6. Поняття, якими у межах шкали визначаються ступені чи інтенсивність зміни станів оцінюваного фактору - оцінювальні показники - повинні бути однорідними, тобто мати спільну природу їхнього визначення та ту ж саму логіку градації.

5.2 Методи багатомірного шкалування

В основу методу покладено ідею відображення певних об'єктів у вигляді точок деякого координатного простору, що має невисоку розмірність. Вихідною інформацією для реалізації методу багатомірного шкалування слугують дані про зв'язки й подібність об'єктів досліджуваної системи.

У межах створених таким чином координат віддзеркалюють структуру системи, зв'язки й подібність об'єктів якої відображуються відстанями між точками. Така проста геометрична модель робить можливим пошук змістовно-інтерпретованого рішення, оскільки осі побудованого простору здатні нести цілком визначене змістове навантаження. Вони можуть бути інтерпретовані як ознаки, покладені в основу зв'язків між об'єктами, або як певний ряд чи шкала ступенів, які віддзеркалюють зміни стану або ознак одного об'єкта, що розглядається за своєю сутністю як складна система.

У першому випадку, характеризуючи кожний об'єкт тими чи іншими ознаками, ми набуваємо можливості отримувати уявлення про певну систему цих об'єктів. У другому випадку метод багатомірного шкалування дозволяє робити висновки про поведінку системи шляхом аналізу комплексного впливу факторів, які відіграють роль причин, на наслідок, що відбиває зміни стану об'єкта під впливом цих причин. За умов, коли інформація отримана експертним шляхом, багатомірне шкалування надає формальний апарат для вивчення поведінки експертів, дослідження питань про те, чим керуються індивіди, які приймають те чи інше рішення.

Багатомірне шкалування застосовується під час обробки й аналізу емпіричних даних або інформації про складні об'єкти чи системи. Зазначена

інформація може бути отримана шляхом опитування експертів або носити характер даних об'єктивних вимірювань.

Мета багатомірного шкалування полягає у виявленні структури та зв'язків досліджуваної множини об'єктів. Під виявленням структури розуміється виділення набору основних ознак або факторів, за якими розрізняються об'єкти, й описанні кожного з об'єктів в термінах цих факторів.

Методами багатомірного шкалування найбільш доцільно користуватися для аналізу експериментальних даних на попередніх стадіях їхньої обробки, а також для аналізу результатів попередніх (пілотних, пошукових) експериментів.

Процедура багатомірного шкалування охоплює ряд послідовних етапів:

Перший етап включає постановку мети шкалування та вибір множини досліджуваних об'єктів. Вирішується питання, які ознаки об'єктів або

ступені змін стану одного об'єкта можуть містити в собі необхідну інформацію та як цю інформацію отримати. На підставі цього формують відповідні матриці експертних оцінок, у боковині яких вказують причини, фактори або ступені їхніх змін, а у головці - наслідок (за типом матриці, наведеної в табл. 5.5).

На другому етапі вирішується формальне завданні побудови координатного простору й розташування в ньому точок - об'єктів - таким чином, щоб відстані між ними, визначені за введеними шкалами, відповідали вихідним розходженням ознак цих об'єктів. Використовуючи координати, ми будуємо геометричне уявлення об'єктів в просторі невеликої кількості вимірювань. При цьому об'єкти, що мають більший ступінь розходжень, повинні знаходитися далеко один від одного, а об'єкти, яким притаманний малий ступінь розходження, - близько.

Методи багатомірного шкалування використовуються для аналізу багатомірних даних, тобто аналізу системи об'єктів або окремих об'єктів як систем, що характеризуються великою кількістю факторів.

Формальним критерієм адекватності може слугувати коефіцієнт кореляції між вихідними й результуючими даними, який і має бути достатньо високим. Засобом підвищення точності формального рішення слугує збільшення розмірності простору: чим вищою є розмірність, тим більшою є можливість отримання точного рішення.

На третьому етапі здійснюється змістова інтерпретація отриманого рішення, яка пов'язана з певними труднощами, оскільки має виконуватися лише спеціалістом, добре знайомим із досліджуваним матеріалом. При цьому координатні осі побудованого простору мають набути смислового насичення, тобто бути інтерпретованими як фактори, що визначають розходження між об'єктами.

Розглянемо приклад застосування методу багатомірного шкалування. Для цього уявимо, що на першому етапі реалізації методу дослідник сформулював таку мету аналізу: "Встановити залежність якості виконання учнями 7-го класу практичного завдання на уроці практичного навчання від використання ними теоретичного знання та трудових навичок".

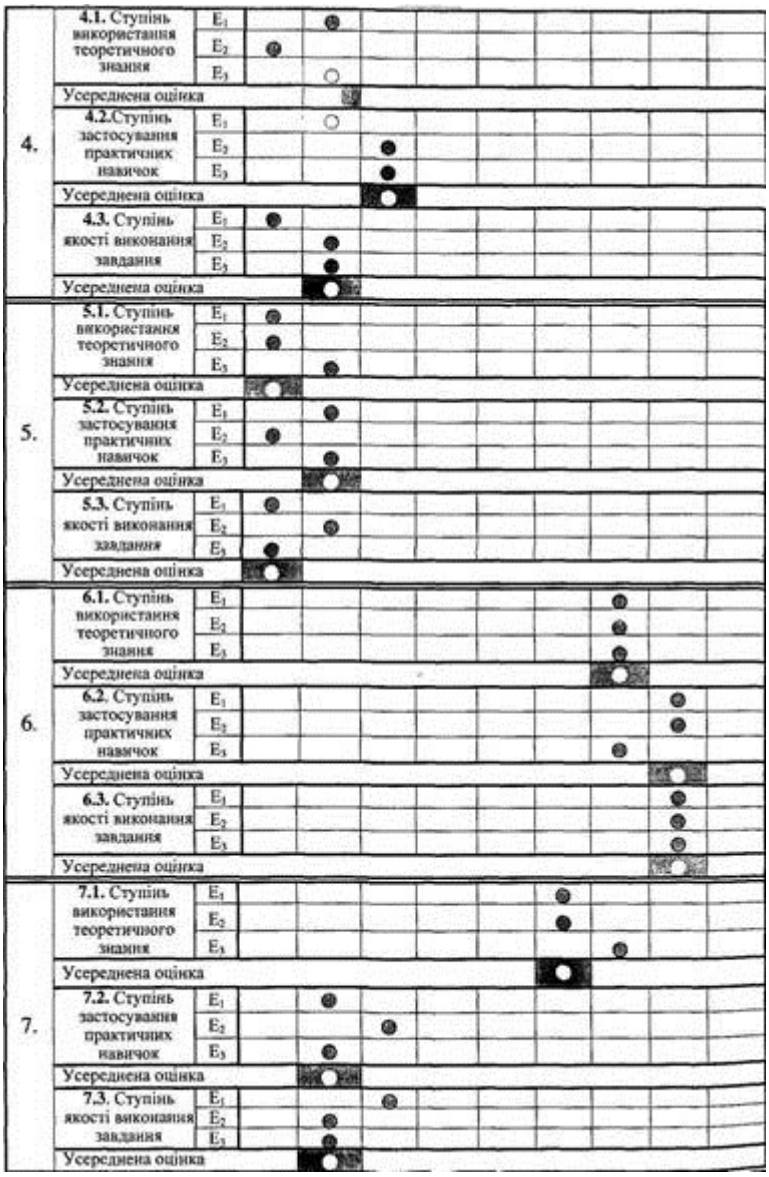
Зрозуміло, що об'єктами дослідження тут мають бути учні. Припустимо, що дослідник передбачає оцінювати їхні дії за трьома ознаками: *ступенем використання теоретичного знання, ступенем застосування практичних навичок та ступенем якості виконання навчально-виробничого завдання.* За цих умов він формує три відповідні шкали експертних оцінок: **"знання", "навички"** та **"якість виконання завдання".** При цьому кількість зон або рівнів змін ступенів прояву оцінюваних ознак приймається ним однаковою для всіх трьох шкал: 1 - найнижчий, 2 - низький, 3 - понижений, 4 - середньопонижений, 5 - середній, 6 - середньопідвищений, 7 - підвищений, 8 - високий, 9 - найвищий. На цих підставах дослідник формує матрицю зведення результатів безпосередніх експертних оцінок, структуру якої відображено в прикладі табл. 8.

У матриці відмічаються рівні або ступені прояву всіх ознак ("знання", "навички", "якість виконаного завдання"), визначені трьома експертами (Еі, E_2, E_3), й наводиться усереднена оцінка цих ознак. Так, якщо в нашому

прикладі (див. табл. 5.4) ступінь застосування практичних навичок учнем, порядковий номер якого відповідно до журналу спостережень зазначено цифрою 1, оцінено експертами Еі та E_2 як "найнижчий", а експертом Ез »[к] "низький", то усереднена оцінка має заноситися в зону "найнижчого" ступеня, відзначеного більшістю (двома з трьох) експертів. Побудовану таким чином матрицю можна аналогічно матриці табл. 5.4 брати за основу якісного опису результатів експертних оцінок, що показано в Табл.5.6.

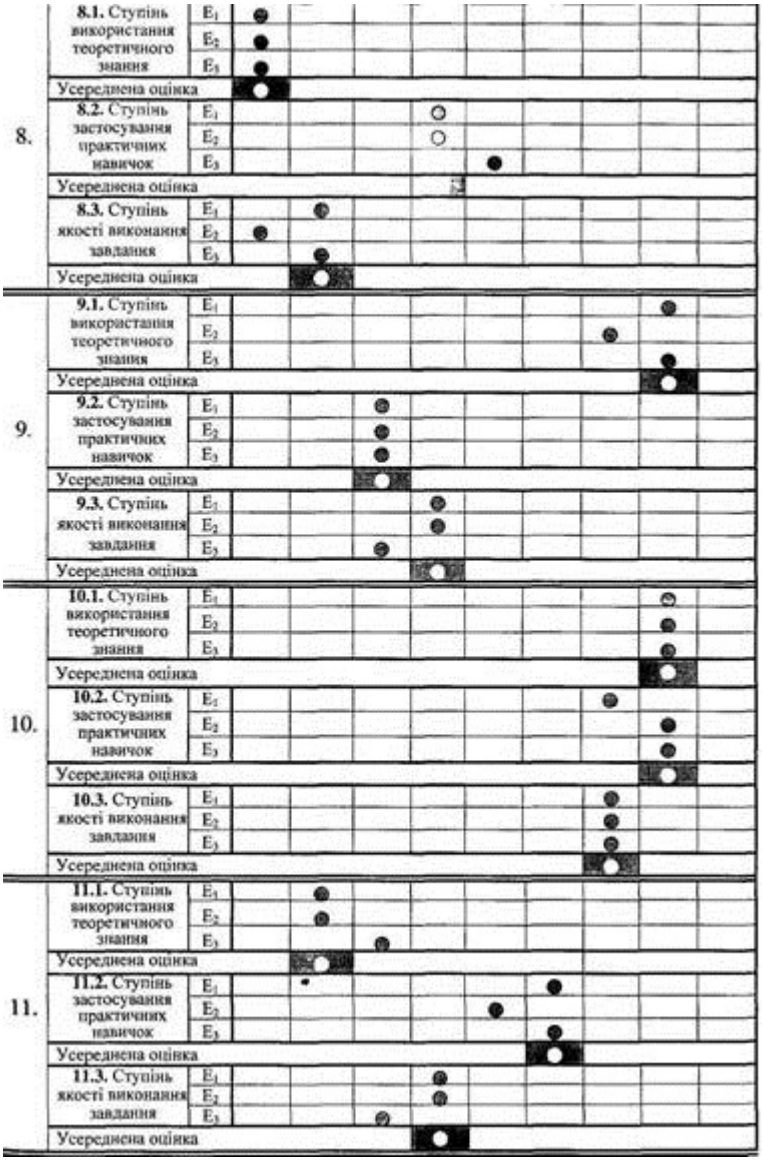
Таблиця 5.6. Матриця зведення результатів безпосередніх експертних оцінок за попередньо заданою шкалою

Порядковий номер учня відповідно до журналу спостережень	Оцінювані показники	Експерти	РІВЕНЬ ПРОЯВУ ОЗНАКИ								
			Найнижчий	Низький	Понижений	Середньо-понижений	Середній	Середньо-підвищений	Підвищений	Високий	Найвищий
			1	2	3	4	5	6	7	8	9
1.	1.1. Ступінь використання теоретичного знання	E_1								●	
		E_2							●		
		E_3							●		
	Усереднена оцінка								○		
	1.2. Ступінь застосування практичних навичок	E_1	●								
		E_2	●								
		E_3		●							
	Усереднена оцінка		○								
	1.3. Ступінь якості виконання завдання	E_1	●								
		E_2		●							
		E_3	●								
	Усереднена оцінка		○								
2.	2.1. Ступінь використання теоретичного знання	E_1					●				
		E_2					●				
		E_3						●			
	Усереднена оцінка						○				
	2.2. Ступінь застосування практичних навичок	E_1	●								
		E_2	●								
		E_3	●								
	Усереднена оцінка		○								
	2.3. Ступінь якості виконання завдання	E_1	●								
		E_2	●								
		E_3	●								
	Усереднена оцінка		○								
	3.1. Ступінь використання теоретичного знання	E_1									●
		E_2									●
		E_3									●
	Усереднена оцінка										○
	3.2. Ступінь застосування практичних навичок	E_1								●	
		E_2									●
		E_3									●
	Усереднена оцінка										○
	3.3. Ступінь якості виконання завдання	E_1								●	
		E_2									●
		E_3									●
	Усереднена оцінка										○



Проте, як зазначалося вище, метод багатомірного шкалування спрямовано на встановлення ступеня кореляції між вихідними даними (факторними показниками) та результуючими показниками. Природно, що за таких умов виникає необхідність використання шкал кількісних оцінок розглядуваних факторів (причин) й результатів їхнього сумісного впливу (наслідків).

Продовження Таблиці 5.6



Тому дослідник має здійснити перетворення шкал якісних показників рівнів прояву ознак у відповідні шкали кількісних показників на основі унормування оцінок, тобто їхнього вираження в долях від одиниці чи відсотках.

Для цього він приймає за 100% "найвищий" рівень прояву оцінюваної ознаки. Тоді, поділивши 100% на дев'ять визначених ним рівнів прояву розглядуваних ознак, дослідник визначає, що діапазон варіювання кожної ознаки в 9-ти рівневій шкалі складатиме 11,11%.

Таким чином, створюється шукана 9-ти рівнева шкала, у межах якої найнижчий рівень прояву ознаки варіює від 0 до 11,11%; низький - від 11,11% до 22,22%; понижений - від 22,22% до 33,33%; середньо-понижений- від 33,33% до 44,44%; середній - від 44,44% до 55,55%; середньопідвищений - від

55,55% до 66,66%; підвищений - від 66,66% до 77,77%; високий - від 77,77% до 88,88% та найвищий - від 88,88% до 100%.

На основі цієї шкали складається матриця кількісних оцінок розглядуваних показників, яка оформлюється у вигляді Табл. 5.7. При її формуванні здійснюються розрахунки усереднених кількісних показників експертних оцінок ступенів прояву факторів, що аналізуються. Для виконання розрахунків використовують результати експертних оцінок, занесені у табл. 5.6.

Таблиця 5.7. Матриця кількісних оцінок ступенів застосування учнями 7-го класу теоретичних знань, практичних навичок та якості виконання ними навчально-виробничого завдання

Порядковий номер учня та прізвище відповідно до журналу спостережень	**Фактори, що оцінюються**	**Кількісни оцінки ступеня прояву розглядуваних факторів**								
		Найнижчий	Низький	Понижений	Середньо-понижений	Середній	Середньо-підвищений	Підви-щений	Високий	Найвищий
		1	2	3	4	5	6	7	8	9
		Інтервал кількісних показників шкали , %								
		0 – 11,11	11,11 – 22,22	22,22 – 33,33	33,33 – 44,44	44,44 – 55,55	55,55 – 66,66	66,66 – 77,77	77,77 – 88,88	88,88 – 100
1. Бурмака П	**1.1.** знання									
	1.2. навички									
	1.3. якість									
2. Бриль М.	**2.1.** знання									
	2.2. навички									
	2.3. якість									
3. Вовк Н.	**3.1.** знання									94,44
	3.2. навички									90,73
	3.3. якість									90,73
4. Гетьман О.	**4.1.** знання									
	4.2. навички									
	4.3. якість									
5. Доренко Ф.	**5.1.** знання									
	5.2. навички									
	5.3. якість									
6. Живило С.	**6.1.** знання									
	6.2. навички									
	6.3. якість									
7. Журавель І.	**7.1.** знання									
	7.2. навички									
	7.3. якість									
8. Косенко В.	**8.1.** знання									
	8.2. навички									
	8.3. якість									
9. Лимар І.	**9.1.** знання									
	9.2. навички									
	9.3. якість									
10. Мазур Т.	**10.1.** знання									
	10.2. навички									
	10.3. якість									
11. Шевченко О.	**11.1.** знання									
	11.2. навички									
	11.3. якість									

Розрахунки усереднених кількісних показників проводять окремо по кожному фактору (у нашому прикладі це "навички, "знання", "результат") за формулою:

$$\overline{G_k^l} = \frac{\sum_1^n q_{min}^n + 0{,}5q \times n}{n}$$, де $\overline{G_k}$ – усереднений кількісний показник експертної оцінки, %; k – порядковий номер об'єкта дослідження (у нашому випадку це порядковий номер учня відповідно до журналу спостережень); l – порядковий номер фактора (в табл. 8 це 1.1, 1.2, 1.3; 2.1, 2.2, 2.3 і т.д.); q_{min}^n – мінімальне значення показника зони шкали кількісних оцінок, яка відповідає якісній оцінці розглядуваного експертами фактора, % (для оцінки "найнижчий рівень" прояву фактора це буде 0%, "низький" рівень - 11,11%, "понижений" - 22,22% і т.д.); n - кількість оцінок, яка має співпадати з кількістю експертів; q - діапазон варіювання кількісної оцінки у межах однієї зони шкали, або ціна поділки шкали, % (для 9-ти рівневої шкали вона складатиме 11,11%).

За результатами проведених у відповідності із зазначеною формулою розрахунків усереднених кількісних оцінок, здійснених трьома експертами, що оцінювали показники 1.1, 1.2 та 1.3 учня із порядковим номером 1 (див. табл. 8), дослідник має отримати:

$$\overline{G_1^{1.1}} = \frac{(0+0+11{,}11)+0{,}5\times 11{,}11\times 3}{3} = 9{,}26\%;$$

$$\overline{G_1^{1.2}} = \frac{(66{,}66+66{,}66+77{,}77)+0{,}5\times 11{,}11\times 3}{3} = 75{,}92\%;$$

$$\overline{G_1^{1.3}} = \frac{(0+11{,}11+0)+0{,}5\times 11{,}11\times 3}{3} = 9{,}26\%;$$

Дії учня із порядковим номером 2 набудуть таких кількісних оцінок:

$$\overline{G_2^{2.1}} = \frac{(0+0+0)+0{,}5\times 11{,}11\times 3}{3} = 5{,}55\%;$$

$$\overline{G_2^{2.2}} = \frac{(44{,}44+44{,}44+55{,}55)+0{,}5\times 11{,}11\times 3}{3} = 53{,}70\%;$$

$$\overline{G_2^{2.3}} = \frac{(0+0+0)+0{,}5\times 11{,}11\times 3}{3} = 5{,}55\%.$$

Подібним чином обчислюються кількісні оцінки й для учня із порядковим номером 3:

$$\overline{G}_3^{3.1} = \frac{(77{,}77 + 88{,}88 + 88{,}88) + 0{,}5 \times 11{,}11 \times 3}{3} = 90{,}73\%;$$

$$\overline{G}_3^{3.2} = \frac{(88{,}88 + 88{,}88 + 88{,}88) + 0{,}5 \times 11{,}11 \times 3}{3} = 94{,}44\%;$$

$$\overline{G}_3^{3.3} = \frac{(77{,}77 + 88{,}88 + 88{,}88) + 0{,}5 \times 11{,}11 \times 3}{3} = 90{,}73\%.$$

Результати подальших обрахувань становитимуть:

$\overline{G}_4^{4.1} = 24{,}07\%$; $\overline{G}_4^{4.2} = 12{,}96\%$; $\overline{G}_4^{4.3} = 12{,}96\%$.

$\overline{G}_5^{5.1} = 12{,}96\%$; $\overline{G}_5^{5.2} = 9{,}26\%$; $\overline{G}_5^{5.3} = 9{,}26\%$;

$\overline{G}_6^{6.1} = 79{,}62\%$; $\overline{G}_6^{6.2} = 72{,}22\%$; $\overline{G}_6^{6.3} = 83{,}32\%$;

$\overline{G}_7^{7.1} = 20{,}37\%$; $\overline{G}_7^{7.2} = 64{,}81\%$; $\overline{G}_7^{7.3} = 20{,}37\%$;

$\overline{G}_8^{8.1} = 42{,}59\%$; $\overline{G}_8^{8.2} = 5{,}55\%$; $\overline{G}_8^{8.3} = 12{,}96\%$;

$\overline{G}_9^{9.1} = 27{,}78\%$; $\overline{G}_9^{9.2} = 79{,}62\%$; $\overline{G}_9^{9.3} = 35{,}18\%$

$\overline{G}_{10}^{10.1} = 79{,}62\%$; $\overline{G}_{10}^{10.2} = 83{,}32\%$; $\overline{G}_{10}^{10.3} = 72{,}22\%$;

$\overline{G}_{11}^{11.1} = 57{,}40\%$; $\overline{G}_{11}^{11.2} = 20{,}37\%$; $\overline{G}_{11}^{11.3} = 35{,}18\%$.

Саме отримані в такий спосіб усереднені кількісні показники оцінок експертами ступенів застосування учнями теоретичних знань, практичних навичок та якості виконання ними навчально-виробничого завдання заносять у табл. 5.7.

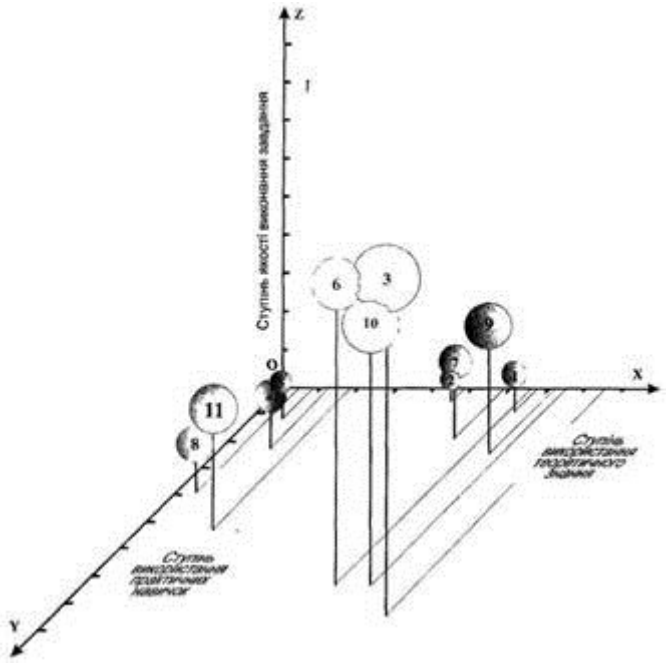


Рис. 5.1. Приклад застосування методу багатомірного шкалування для оцінки впливу теоретичних знань та практичних навичок учнів на якість використання ними практичного завдання

Наведені в таблиці дані, отримані в результаті унормування якісних експертних оцінок, віддзеркалюють зв'язок між оцінюваними параметрами та наслідками, а, отже, складають за своєю суттю певну кількісну модель

складної системи, зображеної на Рис. 5.1. Оскільки експертам було запропоновано однакові шкали оцінок досліджуваних факторів, то координатні осі тримірного простору впливу факторів (причин) на результат (наслідок) були прийняті однаковими, а, отже, їхня максимальна відмітка дорівнюватиме стовідсотковому ступеню прояву кожної із розглядуваних ознак. Для того, щоб отримати змістове направлення осей, яке спрощує інтерпретацію результатів проведених оцінок, у горизонтальній площині факторного простору розташовують вісі ступеня використання теоретичного знання (вісь X) та ступеня використання практичних навичок (вісь Y).Завдяки цьому інтерпретується загальне положення про те, що оволодінню будь-яких практичних навичок має передувати певне теоретичне знання. Така закономірність у загальному вигляді віддзеркалюється залежністю $Y = / (X)$, що означає закріплення за ступенем використання теоретичного знання статусу фактору-причини, зміни якого впливають на ступінь використання практичних навичок, тобто визначають фактор-наслідок. Проте, як на це зверталася увага при постановці завдання багатомірного шкалування, взаємозв'язок цих двох факторів є явищем досить складним і недостатньо вивченим. При цьому, незалежно від питомої ваги кожного з розглядуваних факторів, їх сумісний вплив відбивається у результаті виконання учнем практично-виробничого завдання і може бути оціненим за ступенем якості його виконання. Саме тому шкалу оцінок ступеня якості виконання завдання (або вісь Z) доцільно розташувати перпендикулярно площині XOY. При такому розташуванні будь-яку точку в межах сформованого координатного простору логічно інтерпретувати як наслідок або результат виконання учнем навчально-виробничого завдання, що досягається внаслідок сумісної взаємодії таких факторів, як набуті ним теоретичне знання та практичні навички. Тоді загальна математична залежність для опису розглядуваної нами більш складної функції матиме вигляд $X = F [Y=f(X)]$. Як відомо, про адекватність формування правильного координатного простору свідчить відсутність принципових ускладнень, здатних унеможливити опис результатів дослідження. Отже, за цим критерієм побудований координатний

простір слід вважати адекватним.

Як бачимо, навіть побіжне інтерпретування отриманої таким чином графічної кількісної моделі взаємодії факторів та наслідків дозволяє дійти висновку про те, що результати оцінок створюють три їхні певні групи з чіткою кореляцією вихідних і результуючих даних дослідження. Для більш ретельного змістового аналізу необхідно розглянути проекції точок-результатів на кожну окрему площину, що обмежує простір багатомірного шкалування.

Для подальшого аналізу розглянемо проекції унормованих експертних оцінок на кожну з площин аналізованого простору. У такому випадку, у межах площини, створеної координатами осей ступенів використання учнями практичних навичок і теоретичного знання (Рис. 5.2), ми отримаємо їхнє взаємне розташування, зображене на рис. 5.2.

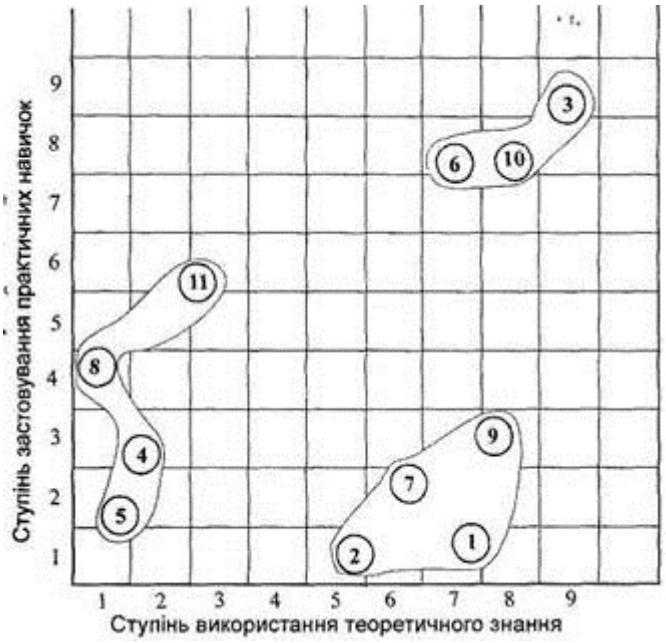


Рис. 5.2. Площина експертних оцінок зв'язку між теоретичним знанням і практичними навичками учнів при виконанні ними практичного завдання

Під час аналізу ми будемо виходити з того, що максимальні показники якості виконання учнями практичного завдання розташовуватимуться в правому верхньому куту зображеної площини. Саме тому відповідно до унормованих показників усереднених експертних оцінок найвищу якість виконання практичного завдання зафіксовано в учнів 3, 6 і 10. Тобто

зазначені показники групуються у певний (виділений нами тонуванням) кластер, ознаками якого є мінімальна віддаль між точками, що відображають зазначену оцінку експертів. Другий кластер, як це видно з рис. 5.2, створюється відповідними показниками оцінки якості виконання практичного завдання учнями 4, 5, 8 і 11. Наступний третій кластер охоплює відповідні показники виконання практичного завдання учнями 1, 2, 7 та 9. Подальшим аналіз дозволяє вважати, що максимальна якість виконання практичного завдання досягається за умов певної рівноваги використання учнем практичних навичок та теоретичного знання під час виконання навчально-практичного завдання (див. кластер 1, точки 3, 6, 10). При цьому превалювання ступеня оволодіння чи застосування учнем практичних навичок над теоретичним знанням здатне суттєво підвищувати якість виконання ним практичного завдання (див. кластер 2, точки 4, 8, 11). На відміну від цього, навіть значне (див. кластер 3, точки 2, 1) переважання теоретичного знання над практичними навичками не здійснює суттєвого впливу на підвищення ступеня якості виконання учнем практичного завдання.

Зіставний аналіз Рис. 5.3 та Рис.5.4 також свідчить, що чітко пропорційне співвіднесення теоретичних знань та практичних навичок, отриманих учнем під час навчально-виробничого практичного навчання, дозволяє виконувати трудове завдання з найвищою якістю (див. кластери 1, точки 3). При цьому певне (у 2 і більше разів) домінування теоретичного знання над практичними навичками або навпаки не призводить до оптимальних показників якості виконання практичного завдання.

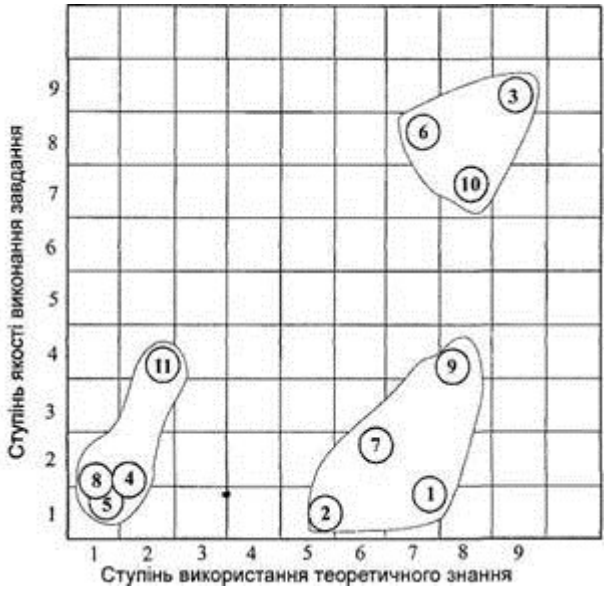


Рисунок 5.3. Площина експертних оцінок впливу теоретичного знання учнів на якість виконання практичного завдання

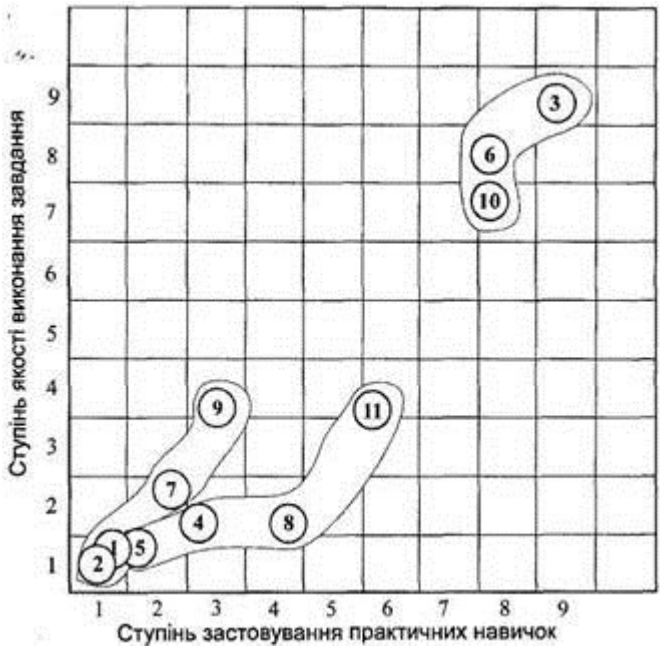


Рисунок 5.4. Площина експертних оцінок впливу практичних навичок учнів на якість виконання практичного завдання

Узагальнюючи результати проведеного аналізу, можна припустити, що, по-перше, максимальна якість виконання практичного завдання учнями має досягатися на основі пропорційного оволодіння ними таким обсягом теоретичного знання, яке є необхідним і достатнім для засвоєння певних практичних навичок. По-друге, оптимальна організація процесу навчання

повинна ґрунтуватися на принципі поступовості - від складного до простого, забезпечуваної пропорційністю засвоєння учнями теоретичних та практичних основ виконання кожного навчально-виробничого завдання.

Зрозуміло, що висунуте на основі методу багатомірного шкалування припущення може виконувати роль робочої гіпотези подальшого експериментального дослідження проблеми взаємодії теоретичного знання та практичних навичок учня як факторів імовірного пливу на ступінь якості виконання навчально-виробничого завдання.

Крім розглядуваних вище методів експертних оцінок та способів їхньої реалізації, існує і ряд інших модифікацій цих методів. Найбільш цікавими є при цьому модифікації, у яких за обраною чи обґрунтованою дослідником шкалою проводиться опосередкована експертиза, що ґрунтується на використанні оцінок, які мають місце в різних наукових джерелах, адекватність яких не викликає сумніву. Такі методи застосовують і для аргументованого доказу теоретичних передумов, і для реалізації опосередкованих емпіричних дій на основі достовірності експериментальних даних, здобутих іншими дослідниками з іншою метою і опублікованих та препарованих ними в будь-якому іншому аспекті.

6 Методи статистичної обробки результатів експериментального дослідження

Статистична обробка та узагальнення результатів експериментальних досліджень спираються на специфічні методи теорії імовірності й математичної статистики, які дозволяють встановлювати ступінь точності й надійності даних вимірювань, здійснюваних в окремих дослідах та їхніх певних серіях.

Динамічні (детермінативні) залежності дозволяють однозначно визначати співвідношення між причиною та наслідком.

Вище нами було наголошено, що процедури обробки та узагальнення

емпіричних результатів дослідження дозволяють знаходити функціональні зв'язки між характеристиками чи параметрами об'єкта, що вивчається. Закономірності або закони, які віддзеркалюють ці зв'язки, прийнято розглядати як певні причинно-наслідкові залежності та відображати їх у вербальному, графічному чи математичному вигляді.

Статистичні залежності не надають можливості однозначного передбачення результату впливу причини на наслідок.

Зазначені закономірності, що описують сутність встановленого шляхом дослідження певного причинно-наслідкового зв'язку між явищами та параметрами окремого об'єкта, поділяють на два їхні типи. Закономірності першого типу, які називаються динамічними (детермінативними, причинними), характеризуються тим, що кожній з описуваних ними причин відповідає цілком конкретний наслідок або кожному значенню параметра, що відіграє роль причини відповідає лише одне чітко визначене значення параметра, який є наслідком. Інакше кажучи, динамічні причинно-наслідкові закономірності підкоряються правилу однозначної відповідності причини та наслідку. Природно тому, що і передбачення наслідків явища, описуваного динамічними закономірностями або законами, є завжди простим та однозначним. Найбільш адекватно динамічні закономірності описують природні фізичні й механічні явища або процеси та відбивають зв'язок параметрів у більшості створюваних людиною технічних засобах.

Закономірності другого типу, які називаються статистичними, також є придатними для опису причинно-наслідкового зв'язку. Проте на відміну від динамічних вони не надають можливості однозначного передбачення наслідку, який виникає під дією конкретної причини. Тому на підставі статистичної залежності передбачення може бути здійснено лише з тим чи іншим ступенем вірогідності. За допомогою статистичних закономірностей та законів прийнято описувати поведінку так званих складних систем: взаємодія космічних тіл, кліматичні зміни, метереологічні явища, хімічні й біохімічні реакції, коливання чисельності біологічних популяцій, поведінка людини чи

певного соціуму, розвиток культури, науки, мови тощо.

Оскільки поведінка складних систем зумовлюється множинною взаємодією факторів різної природи, то наслідки такої взаємодії прийнято називати випадковими. Випадковість тут означає підпорядкованість особливим статистичним імовірнісним закономірностям, прояв яких і розглядає математична статистика.

У межах математичної статистики, основою якої є сучасна теорія імовірності, напрацьовано ефективний інструментарій обробки даних кількісних вимірювань, виконуваних під час проведення експериментальних досліджень. Результати зазначених напрацювань широко відображено у відповідних наукових працях та навчальних посібниках, що мають різні рівні використання математичного апарату, а також значно відрізняються за дохідливістю викладу й придатністю для практичного використання висвітлених у них методів. При цьому матеріали багатьох існуючих джерел містять у собі основні математичні залежності та наочні приклади статистичної обробки результатів досліджень типових об'єктів різних галузей технічної [26; 27; 28; 29; ЗО; 31] та гуманітарної [32; 33; 34; 35; 36] сфер наукового пізнання.

Враховуючи наявність значної кількості наукових джерел, що охоплюють різну за рівням складності інформацію про прикладне застосування теорії імовірності у науковій практиці, ми вважаємо доцільним розглянути лише ряд важливих положень, що висвітлюють сутність найбільш часто вживаних методів статистичної обробки експериментальних даних.

Нагадаємо насамперед, що статистичній обробці підлягають цифрові показники характеристик або параметрів, вимірюваних під час проведення дослідів.

6.1 ОСОБЛИВОСТІ ХАРАКТЕРИСТИК ПАРАМЕТРІВ ВИМІРЮВАННЯ. ВИДИ ПОХИБОК

Вимірюванням будь-якої фізичної величини називають операцію,

внаслідок якої ми дізнаємося у скільки разів вимірювана величина більша (або менша) відповідної величини, прийнятої за еталон [29:5].

При цьому, до завдання вимірювань входить не лише знаходження певної фізичної величини, але також і оцінка допущеної при її вимірюванні похибки.

Практика показує, що жодне вимірювання не може бути виконаним абсолютно точно, оскільки його результати завжди містять деяку помилку. **Вимірювання** здебільшого намагаються проводити якомога точніше, тобто зробити його похибку по можливості малою. Проте слід пам'ятати, що чим точніше ми бажаємо вимірювати, тим важче це робити. Тому не слід виконувати вимірювання із точністю більшою, ніж та, що необхідна для вирішення поставленого завдання (наприклад, немає сенсу вимірювати довжину будівлі з точністю до міліметру).

Очевидно, що, вимірюючи за допомогою будь-якого інструменту певну величину або параметр, ми, як правило, не можемо зробити похибку меншою, ніж та, яка визначається похибкою вимірювального пристрою.

У зв'язку з тим, що ніяке вимірювання не може бути виконане абсолютно точно, у дослідницькій практиці прийнято розрізняти три основних типи похибок:

1) **Систематичні,** величина яких однакова в усіх вимірюваннях, що проводяться тим самим методом, тими самими приладами або інструментами.

Іншими словами, систематичні похибки породжуються при багаторазовому повторенні тих самих вимірювань (така похибка є здебільшого похибкою самого вимірювального приладу або інструменту).

2) **Випадкові,** величина яких є різною навіть для вимірювань, ' виконуваних однаковим чином.

Сутність випадкових похибок полягає в тому, що на їхню появу впливає ряд факторів або причин, дія яких не є однаковою під час кожного повторного вимірювання, і тому не може бути передбаченою (такими причинами можуть слугувати зміни, що відбуваються в самому об'єкті, та

зміни у вимірювальних приладах, які зумовлюються відповідними змінами оточуючого середовища: коливань повітря, різниці в температурі об'єкта й приладу, електромагнітний вплив тощо).

3) Промахи або грубі похибки, джерелом виникнення яких завжди є недостатня увага того, хто виконує вимірювання.

Причиною виникнення грубих похибок може бути: нечітке градуювання шкали вимірювального приладу, що призводить до невірного зчитування її показників, невірний запис показників вимірювального приладу в журнал спостережень тощо.

Особливу увагу під час проведення вимірювань слід звертати на виявлення та виключення ***систематичних похибок,*** величина яких у ряді випадків може бути на стільки значною, що призведе до викривлення результатів вимірювань.

Систематичні похибки прийнято поділяти на чотири групи [29:14-20]:

1) похибка, природа якої досліднику відома, а її величина може бути визначена достатньо точно. Такі похибки називають **поправками** (наприклад, під час вимірювання довжини латунного стержня сталевим штангенциркулем при температурі оточуючого середовища яка дорівнює 30°С слід уводити поправку на різницю їхнього температурного розширення, тобто на зменшення вимірюваної величини, яке складатиме 0,03 мм).

2) похибка, походження якої нам відоме, а її велична - ні.

Такою похибкою може бути погрішність вимірювального приладу, яка визначається класом його точності (наприклад, клас точності 0,5 означає, що показання приладу є правильними з точністю до 0,5% від максимального значення вимірюваного ним параметра).

3) похибка, про існування якої ми не здогадуємося.

Така похибка найчастіше зустрічається під час складних вимірювань (наприклад, при вимірюванні густоти зразка будь-якого металу шляхом віднесення ваги зразка до його об'єму ми можемо зробити грубу похибку, якщо в середині цього зразка є порожнини або повітря, яке потрапило туди в ливарному процесі).

4) похибка, природа якої зумовлена властивостями вимірюваного об'єкта (наприклад, така похибка має місце при одноразовому вимірюванні діаметру циліндра, що має певну ступінь еліптичності).

Слід пам'ятати, що за певних обставин систематична похибка може бути переведена у випадкову. Для цього здебільшого буває достатнім здійснити декілька вимірювань і взяти середнє арифметичне. Таке переведення систематичних похибок у випадкові часто виявляється корисним, оскільки воно дозволяє покращити точність вимірюваних результатів.

Отже, якщо виконати ряд вимірювань та обрахувати середнє арифметичне з цього ряду, то випадкова похибка середнього арифметичного буде меншою за похибку одиночного виміру. Тому для зменшення випадкової похибки слід провести не одне, а ряд вимірювань. При цьому, чим більше буде зроблено вимірювань, тим меншу величину випадкової похибки ми отримаємо. Проте не слід забувати, що немає сенсу проводити вимірів більше, ніж це необхідно для того, щоб систематична похибка суттєво перевищувала випадкову.

На викладеному ґрунтуються такі правила [29:20-21]:

- якщо визначальною є систематична похибка (тобто її величина є суттєво більшою за величину випадкової похибки, притаманної цьому методу вимірювання), то достатньо виконати вимірювання один раз;
- якщо визначальною є випадкова похибка, то вимірювання слід проводити декілька разів. Число вимірювань доцільно вибирати таким, щоб випадкова похибка середнього арифметичного була меншою за систематичну похибку і з тим, щоб остання знову визначала кінцеву похибку результату.

При цьому, слід звернути увагу на те, що одним вимірюванням можна обмежитися лише у тому випадку, коли з будь-яких джерел нам відомо, що величина випадкової похибки меша за систематичну.

Така ситуація має місце, зазвичай, тоді, коли вимірювання проводяться

відомим методом, похибки якого певною мірою вивчені (наприклад, якщо вимірювати довжину олівця за допомогою вимірювальної лінійки з похибкою поділок в 1мм, то можна бути впевненим, що випадкова похибка буде набагато меншою за 1мм і слід обмежитися одним вимірюванням).

Подібним чином, якщо ми точно знаємо, що похибка зважування на торгових вагах менша за ціну їхньої поділки, яка дорівнює 5г, то зважувати на них необхідно лише один раз.

Як бачимо, ***необхідне число вимірювань залежить від співвідношення величин систематичної й відносної похибок.***

Зазначимо тут, що незалежно від походження похибок їхні кількісні оцінки виражаються в абсолютних величинах, які мають розмірність вимірюваного параметру. Таку величину називають абсолютною похибкою та позначають символом $_д\%$. Проте при безсумнівній для людини зручності застосування абсолютної похибки у практично-перетворювальній діяльності у разі необхідності зіставлення точності різних вимірювань виникає потреба у використанні відносної похибки як основної характеристики якості виконуваних вимірювань.

Відносною похибкою називають відношення абсолютної похибки ($_{\Delta}\chi$) до вимірюваної величини (x) виражене у відсотках:

$$_{\Delta}\chi = \frac{\Delta\chi}{\chi} \times 100\%.$$

Необхідність обчислення відносної похибки пов'язана з тим, що вказівка на абсолютну похибку вимірювань мало говорить про дійсну точність, якщо не зіставляти її з самою вимірюваною величиною.

Випадковими називають такі події, про появу яких не може бути зроблене точне передбачення.

Для аналізу якісної сторони вимірювань, що містять випадкові похибки, здійснюють відповідну статистичну обробку отриманих результатів, яка дозволяє встановити їхню точність та надійність.

Вище згадувалося, що, по-перше, у тих випадках коли при вимірюваннях фізичних величин основну роль відіграють випадкові похибки,

будь-які оцінки точності вимірювань можна робити лише з деякою ймовірністю. По-друге, для того, щоб виявити випадкову похибку вимірювань необхідно повторювати їх декілька разів. При цьому, якщо кожне вимірювання дає дещо відмінні від інших вимірювань результати, ми маємо справу із ситуацією, коли випадкова похибка відіграє суттєву роль.

Саме в таких випадках результати дослідів прийнято обробляти на основі методів теорії імовірності та математичної статистики, що дозволяють здійснювати елементарні оцінки похибок вимірювань.

В експериментальних розділах наукових праць методи математичної статистики застосовуються на різних стадіях дослідження для виконання таких процедур, як виявлення грубих похибок (промахів) результатів вимірювання, обробка узагальнення результатів, планування необхідної точності та кількості вимірювань, а також планування багатофакторних експериментів.

6.2 Основні положення теорії похибок

У загальному випадку результати експериментальних вимірювань складають певну статистичну сукупність вихідних даних *(x_u x_2, хз, ... x_n,* х,,), які відбивають зміну

Сукупністю прийнято називати множину подібних об'єктів вимірюваного показника в часі або характеризують його причинно-наслідкову зміну в залежності від відповідної зміни іншого показника чи фактора, що відіграє роль причини.

Під **розподілом випадкової величини** розуміють сукупність усіх можливих значень випадкових величин і відповідних вірогідностей їхньої появи.

Оскільки, як зазначалося вище, результати, отримані в експерименті, залежать від цілого ряду факторів, то вони, як правило, є випадковими величинами, що підкоряються дії закону їхнього нормального розподілу.

Законом розподілу випадкової величини називають співвідношення, яке

встановлює зв'язок між можливим значенням випадкової величини та вірогідністю її появи у процесі виконання вимірювань.

Практика вимірювань показала, що випадкова величина буде розподілятися за нормальним законом у тих випадках, коли вона являтиме собою суму дуже великого числа взаємно незалежних випадкових величин, вплив яких на зазначену суму є надзвичайно малим.

Випадкова величина може бути дискретною (вимірюваною через певні проміжки часу) та неперервною. Неперервна випадкова величина приймає будь-яке значення з інтервалу свого вимірювання, а дискретна - лише певні значення.

Припустимо, що було виконано серію з ***n*** експериментів. При цьому результат вимірювань проведених у межах їх даної серії як безперервну випадкову подію (де) було зафіксовано в певному інтервалі к раз. У такому випадку **частоту випадкової події** прийнято характеризувати відповідним відношенням. З теорії імовірності відомо, що при збільшенні ***n*** відношення *Пііп* приймає все більш стійке значення, тобто стає статистично стійким.

Межу, до якої прагне наблизитися відношення *njn* при необмеженому збільшенні кількості експериментів я, називають **вірогідністю випадкової події *x.*** У математичному вигляді його записують так:

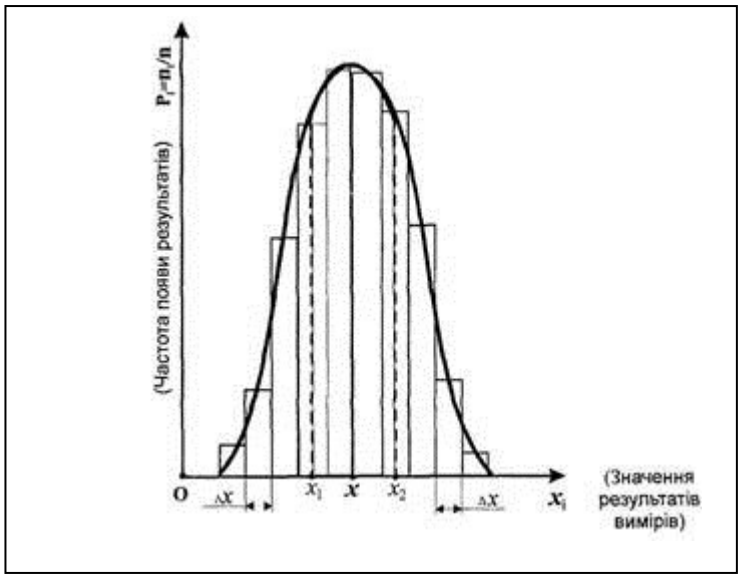
$$P_{(x)} = \lim_{n\to\infty} n_i/n .$$

Вірогідність - числова характеристика ступеня можливості появи будь-якої випадкової події при тих або інших певних умовах, що можуть повторюватися необмежену кількість разів [4:211].

Сукупність усіх можливих значень випадкової величини в розглядуваних вище умовах прийнято називати **генеральною сукупністю.**

Деяку сукупність результатів вимірювань в експериментах, що має місце у дійсних умовах, називають **вибіркою.** При цьому число експериментів або результатів вимірювань, з яких складається вибірка, представляє її об'єм.

Результати серії вимірювань однієї випадкової величини можна наочно відобразити у вигляді гістограми, яка показує, як часто були отримані ті чи інші значення вимірюваної величини. Гістограма будується шляхом розподілу всього діапазону вимірюваних значень на рівні інтервали й підрахунку частоти появи вимірюваної величини у кожний інтервал.



Для цього [27:140-145; 36:149-150] отримані результати розташовують у вигляді ряду по збільшенню бо зменшенню їхньої абсолютної величини та групують за інтервалами (Рис. 6.1).

Рисунок 6.1 – Структура визначеної кривої розподілу з даних експериментів

Кількість інтервалів визначається за наближеною залежністю:

£ = 1 + 3,2 xlg/?,

де *к* - кількість інтервалів, *n* - число вимірів.

Отримане значення кількості інтервалів округляють до його найближчого значення.

Ширина інтервалу розраховується за формулою:

$$d = \frac{x_{max} - x_{min}}{k},$$

де d – ширина інтервалу, x_{max} та x_{min} – максимальне та мінімальне значення отриманих результатів.

Далі на горизонтальній осі відмічають межі інтервалів та над кожним з них будують прямокутник, основа якого дорівнює ширині інтервалу (дХ = *d)* та висотою, що дорівнює відношенню *n/n,* де *ni* - число вимірювань, які входять до даного інтервалу, *an* — загальне кількість вимірювань, здійснених у їхній аналізованій серії.

Як бачимо, на гістограмі вимірювані (рис. 6.1) значення випадкової

величини розташовуються більш-менш симетрично навколо істинного значення, відміченого на рисунку символом *x*. Крім того, очевидно, що великі відхилення випадкової величини від цього значення зустрічаються на гістограмі рідше, ніж малі.

Навіть при обмеженій кількості вимірювань на гістограмі можна провести плавну криву, яка пройде через середні точки вершин прямокутників, тобто через середні значення результатів вимірювань, здійснених у межах кожного окремого інтервалу.

Ця крива й визначає в основі своїй форму закону розподілу результатів вимірювань випадкових величин.

Для побудови більш точної кривої необхідно значно збільшити число вимірювань у межах їх розглядуваної серії. Унаслідок збільшення числа отриманих результатів збільшується також і кількість інтервалів, а, отже, й

зменшиться їхня основа Δx.

Якщо за таких умов (збільшення числа експериментальних вимірів n та зменшення ширини інтервалу Δx) за віссю ординат відкладати відношення $P_i = n/n$, то при $\Delta x \to 0$ вершини прямокутників зіллються в плавну лінію,

котру й прийнято називати кривою розподілу результатів експериментів. (рис. 6.1).

Ордината цієї кривої буде являти собою щільність розподілу вірогідностей випадкової величини, що знаходиться в інтервалі від *x* до *x*+*dx*. Математично це буде записуватися так:

$$P_{(x)}\,dx = P\,\{x \le \sum \le (x+dx)\}.$$

Якщо на горизонтальній осі виокремити довільний інтервал (x_1, x_2), то затонована на рис. 42 площа, яка розташована під кривою в цьому інтервалі, дорівнюватиме вірогідності того, що вимірювана величина або показник знаходяться в даному інтервалі. Саме з цим і пов'язано вживання поняття – щільність розподілу вірогідності, яке у математичному інтерпретуванні набуває такого вигляду:

$$F_{(x)} = \int_{x_1}^{x_2} P_{(x)}dx,$$

де $F_{(x)}$ – функція розподілу вірогідності випадкової величини x.

Зазначена функція зв'язана зі щільністю розподілу вірогідностей таким диференціальним рівнянням:

$$P_{(x)} = \frac{dF_{(x)}}{dx}.$$

Звернемо увагу на різницю, що існує між гістограмою розподілу результатів вимірюваних величин та кривою розподілу цих результатів (рис. 42). Якщо гістограма описує конкретну серію експериментів, то на відміну від неї, крива розподілу віддзеркалює сукупність уявного безкінечного числа вимірювань такої величини даним методом. Таким чином, крива розподілу завдяки її теоретичної природи набуває здатності найбільш повно відображати умови експерименту та характеризувати якісний бік вимірювань.

Скориставшись цією відмінністю, розглянемо ряд особливостей, притаманних даній кривій. Для зручності розгляду виокремимо криву розподілу, наведену на рис. 6.1, та відтворимо її в більш наочних координатах (Рис. 6.2).

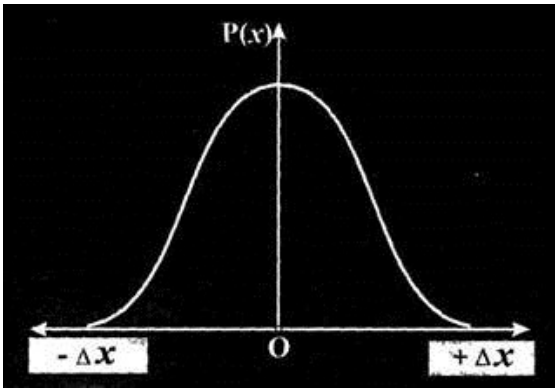


Рисунок 6.2. Крива щільності нормального розподілу результатів [1,
вимірювання випадкових величин

Центром таких координат слугуватиме точка О, яка відповідає точці *x* (рис. 6.1) та означає істинне значення вимірюваної експериментальної величини. Тоді ліворуч від початку нових відносних щодо точки О координат розташовується вісь, яка відображатиме похибки вимірювань із знаком мінус (-дх), а праворуч - вісь похибок, що мають знак плюс (+дх). Вісь ординат Р^ відбиватиме показники щільності розподілу вірогідності появи тієї чи іншої похибки.

Таким чином, на рис. 6.2 отримуємо загальний вигляд теоретичної кривої щільності нормального розподілу результатів вимірювання випадкових величин, яку прийнято називати кривою Гауса. У зв'язку з цим,

найбільш часто вживаний у математичній статистиці закон теорії імовірності та нормального розподілу випадкових величин називають нормальним законом Гауса.

Формула Гауса неодноразово була перевірена експериментально, унаслідок чого встановлено, що там, де похибки не є занадто великими, вона відмінно узгоджується з результатами вимірювань

В основу математичного доведення нормального закону розподілу похибок (або формули Гауса) було покладено такі припущення:

1) похибки вимірювань можуть приймати безперервний ряд значень;

2) при великій кількості вимірювань похибки однакової величини, але різні за знаком будуть зустрічатися однаково часто;

3) частота появи похибок зменшуватиметься із збільшенням величини похибки (тобто великі похибки мають спостерігатися рідше, ніж малі).

Згідно із законом великих чисел при нескінченно великій кількості вимірювань n істинне значення вимірюваної величини x має дорівнювати середньоарифметичному значенню всіх результатів вимірювань x .

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i ,$$

де $\bar{x}$ – середнє арифметичне значення вимірюваної величини (або середнє значення), n – кількість вимірювань, x_i – значення конкретного вимірювання.

Похибку конкретного вимірювання прийнято записувати у вигляді:

$$\Delta x_i = x_i - x,$$

де x – істинне значення вимірюваної величини.

На підставі цього можна записати вираз для похибки n повторностей вимірювань.

$$\sum_{i=1}^{n} \Delta x_i = \sum_{i=1}^{n} x_i - nx .$$

Якщо кількість вимірювань n є великою ($n \to \infty$), то набуває чинності така рівність:

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} \Delta x_i = 0$$

Таким чином, при нескінченно великій кількості вимірювань істинне значення вимірюваної величини x стає рівним середньоарифметичному

значенню x усіх результатів вимірювання, тобто:

$$x = \bar{x} \text{ при } n \to \infty .$$

У разі обмеженої кількості вимірювань $(n \neq \infty)$ середньоарифметичне значеннях відрізнятиметься від істинного значення, а попередня рівність стане наближеним і набуде вигляду;

$$x \approx \bar{x} .$$

За таких умов, які виникають неминуче в експериментальній практиці, і постає необхідність в оцінці зазначеного розходження.

Зрозуміло, що якщо ми матимемо в наявності лише один результат вимірювань Хі, то оцінка істинного значення вимірюваної величини буде менш точною, ніж середньоарифметична оцінка при будь-якій кількості повторних вимірювань:

$$|x - \bar{x}| < |x - x_i|.$$

Поява такого чи іншого значення x_i під час вимірювання є подією випадковою. Зважаючи на це, позначимо вірогідність появи x_i в інтервалі від x_1 до x у такому вигляді:

$$P_1 (x_1 < x_i < x).$$

Відповідним чином вірогідність появи результату x_i в інтервалі від x_2 до x позначимо виразом:

$$P_2 (x_2 < x_i < x).$$

Тоді, якщо $x_1 - x_2 = dx$, прирощення вірогідності може бути відображено таким чином:

$$dP = P_1 (x_2 + dx < x_i < x) - P_2 (x_2 < x_i < x) = f (x_i)\, dx.$$

За таких умов $\frac{dP}{dx} = f(x_i)$ буде тим більшою, чим менше абсолютне відхилення.

Величину $\frac{dP}{dx}$, як це зазначалося вище (див. рис. 1), прийнято називати щільністю розподілу випадкової величини $P_{(x)}$:

$$P_{(x)} = \frac{dP}{dx}.$$

Звернемо увагу на те, що функція $P_{(x)}$ характерна тим, що добуток $P(x)\, dx$ *дає* частку повного числа результатів вимірювань n, які припадають на інтервал від x до $x+dx$.

Іншими словами, добуток $P_{(x)}\, dx$ *є* вірогідністю того, що окремо випадково вибране значення вимірюваної величини x, знаходитиметься в інтервалі від x до $x+dx$.

Саме ця вірогідність, як про це свідчить теорія вірогідності,

визначається законом нормального розподілу Гауса.

Згідно з цією формулою, функція *щільності* нормального розподілу випадкової величини характеризується двома параметрами: істинним значенням вимірюваної величини *x* та середньоквадратичним відхиленням σ.

Крива Гауса свідчить, що щільність нормального розподілу випадкової величини є симетричною відносно точки О, тобто - симетричною відносно істинного значення вимірюваної величини *x*.

При цьому щільність нормального розподілу досягає максимуму саме в цій точці при $X_i = x$ та наближається до нуля при збільшенні похибки конкретного вимірювання.

Для підвищення чіткості подальшого викладу звернемо увагу на таке.

По-перше, (для документів, рукописів або встановлення фактів). У разі вивчення станів складних фізичних об'єктів чи предметів доцільно використовувати координатну логіку вибору вимірів, із яких і складатиметься генеральна сукупність отриманих результатів.

Вибірковою сукупністю, або вибіркою, називають певну частину множини об'єктів, що складають генеральну сукупність.

Використання вибірок у статистиці зумовлено практичною неможливістю дослідження всіх об'єктів генеральної сукупності.

По-друге, межі вибору будь-якої генеральної сукупності вимірюваних параметрів чи факторів або розглядуваних ознак визначаються дослідником виключно в суровій відповідності до мети започаткованого експерименту. При цьому у випадках проведення складних експериментів, що супроводжуються необхідністю поділу генеральної сукупності на певні вибіркові сукупності (або генеральної сукупності вимірів на певну кількість їхніх класів чи підкласів), дії дослідника в межах мети експерименту мають бути співвіднесеними ще й із завданнями та окремими питаннями експериментального дослідження.

Генеральною сукупністю називають об'єкти чи явища, об'єднані за певними кількісними або якісними ознаками.

По-третє, репрезентативність вибірок та підвибірок або класів і підкласів відповідних вимірювань прогнозується за принципом адекватності досліджуваному явищу чи процесу. Цим принципом зумовлюється правило вибору об'єму репрезентативної вибірки, відповідно до якого, чим нижчою є частота явища, що вивчається, тим більшим має бути обсяг репрезентативних вибірок або підвибірок при тій же самій точності результатів.

Прийнято розрізняти структуровану та неструктуровану вибірки. Структурна вибірка складається з ряду підвибірок. Неструктурована вибірка на підвибірки не членується.

Повертаючись до наведеної вище теоретичної формули Гауса, яка описує щільність нормального розподілу випадкової величини, розглянемо сутність статистичного поняття дисперсії.

Дисперсією випадкової величини називають середнє значення квадрата відхилення випадкової величини від її середньоарифметичного значення.

Виходячи з викладеного вище, неважко зрозуміти, що процедура статистичної обробки даних експериментального дослідження за критерієм Стьюдента також може бути представлена в алгоритмічному вигляді.

Згідно з алгоритмом статистичні показники результатів дослідження обчислюються в такій послідовності.

1. На підставі зведених у таблиці результатів окремих вимірів оцінюваного параметра визначають його середнє значення.

2. Проводять порівняння фактичного й табличного критеріїв Стьюдента. Табличний критерій Стьюдента визначається за кількістю здійснюваних вимірів оцінюваного параметра при обраній довірчій ймовірності $\alpha = 0{,}95$. Після цього у випадку $t_ф > t_T$ здійснюють вилучення грубої помилки та повторну обробку решти значень. У разі $t_ф \le t_T$ продовжують обчислення й визначають похибку середнього

результату

Отримані таким чином значення випадкової похибки вимірювань фізичних показників задають довірчим інтервалом і довірчою ймовірністю, які наводять у відповідних таблицях, що при необхідності містяться у додатках до наукової праці або дисертації. Інші результати, отримані в процесі експериментального дослідження, та існуючі між ними залежності викладають у формі таблиць, діаграм тощо.

Нагадаємо також, що обробку кількісних показників, отриманих у вигляді цифрових даних за результатами багаторазових експериментальних вимірювань фізичних величин, проводять у відповідності із відомими методами теорії ймовірності та математичної статистики, розмаїття прикладів практичного застосування яких розглянуто в джерелах.

7 Емпіричні методи дослідження декомпозиції програмних систем, зв'язаності і зчепленості їх компонентів

7.1 Декомпозиція підсистем на модулі

Відомо два типи моделей модульної декомпозиції:

- Модель потоку даних;
- Модель об'єктів.

В основі моделі потоку даних лежить поділ за функціями програмної системи.

Модель об'єктів базується на слабо щеплених сутностях, які мають власні набори даних, стани і набори операцій.

Вибір типу декомпозиції визначається складністю підсистеми, яку поділяють на модулі.

Розділення програми на модулі – один з ефективних шляхів управління складністю і спосіб побудови структурних програм. Ідеї модульного програмування пов'язані з принципами „розділяй і володарюй" і ієрархічного впорядкування. Основні цілі модульного програмування:

Декомпозиція програми на незалежні фрагменти – модулі, які можуть

бути надалі реалізовані у вигляді окремих процедур або функцій, або у вигляді модулів, що є конструкціями конкретної мови програмування.

Розділення складної проблеми на більш дрібні і прості підпроблеми.

Незалежна розробка і тестування кожного модуля.

Модульне програмування – це організація програми у вигляді невеликих незалежних блоків, які називають модулями, структура і поведінка яких визначається деякою сукупністю правил. Таким чином, модуль – основний будівельний блок структурної програми. Кожний модуль повинен відповідати одній проблемно-орієнтованій задачі.

Переваги модульного програмування:

Модульна програма простіше для розуміння.

Спрощено тестування програми.

Спрощений процес виявлення і виправлення помилок.

Зміни, що вносяться в програму, можуть бути обмежені небагатьма модулями.

Спрощений процес підвищення ефективності програми.

Окремі модулі можуть повторно використовуватися в інших програмах.

Час розробки складної програми може бути помітно скорочено, оскільки розробка окремих модулів може бути доручена різним програмістам.

7.2 Особливості структурних програм

Становлення структурної методології пов’язане з появою ряду концепцій в галузі програмування, що виникли через зростання складність вирішуваних проблем. Однією з перших з’явилася ідея низхідного програмування (проектування), або покрокова деталізація, яка має на увазі розбиття складної задачі на декілька підзадач. Таке послідовне розбиття продовжується до тих пір, поки отримані підзадачі не стануть настільки простими, що їх можна легко записати мовою програмування. В результаті послідовної деталізації програму можна представити у вигляді ієрархічної структури взаємозв’язаних модулів відповідно до розподілу підзадач за

рівнями ієрархії. Проте побудова програми можливо і з низу до верху – від конкретних об’єктів і функцій конкретної машини до більш абстрактних об’єктів і функцій, які використовуються в модулях програми. Цей підхід отримав назву рівнів абстракції.

Використання в програмуванні перерахованих підходів приводить до створення структурних програм, що мають ряд особливостей. Найістотнішими характеристиками структурної програми є її ієрархічна форма і використання обмеженого набору управляючих конструкцій. Важливим представляється також проста стандартна структура управління, єдині вимоги до документування програми та узгодженість про стиль кодування.

Структурна програма розділяється на ряд модулів, впорядкованих ієрархічно відповідно до їх логічних зв’язків і зв’язків за послідовністю виконання. Базовий набір управляючих структур обмежений трьома базовими конструкціями: послідовність, вибір, повторення. Для розширення можливостей прийняття багатоальтернативних рішень базовий набір доповнений оператором багатоальтернативного вибору. В структурних програмах допускається використання оператора переходу (GO TO), але тільки для переходу в точку виходу даного модуля.

Будь-яка структурна програма, утворена з базових конструкцій, має властивості модульності, і невід’ємною її частиною є документація, яка забезпечує розуміння програми. Прийнято використовувати три рівні документування:

Загальний огляд програми;

Організація програми;

Оператори програми.

Документування звичайно проводять в вихідному коді програми для опису загальних функцій програми, для представлення функцій кожного модуля, для опису даних і параметрів, які передаються між модулями. Разом з внутрішньою може бути і зовнішня документація. Дуже часто це графічне представлення структури програми, логіки обробки, структур даних.

Документація загального огляду поміщається на початку вихідного тексту програми у вигляді коментарів. До оглядової інформації відносяться:

Короткий опис загальної функції програми, що визначає її основні компоненти.

Короткий опис бази даних, включаючи головні файли і структури записів.

Стислий виклад ідей, що використовуються в проекті, стиль програмування і т.п.

Посилання на попередню документацію, проектні документи і матеріали.

Посилання на програмну документацію, яка забезпечує експлуатацію програми.

Елементи блоку коментарів для кожного модуля містять:

Призначення модуля (в одному – двох рядках);

Алгоритмічні особливості, обмеження;

Опис інтерфейсів модуля (параметри);

Реакція на помилки і процедури відновлення при помилках;

Інформація про вплив змін в модулі на інші частини програми.

Коментарі до операторів використовуються тільки у виняткових випадках для пояснення незвичайних або складних алгоритмічних конструкцій. Замість коментарів часто використовуються змістовні імена процедур і змінних, типові управляючі конструкції і процедури, послідовний, строгий стиль кодування.

Стандартизація стилю кодування передбачати виконання наступних правил.

В кожному рядку записується один оператор. Якщо для запису потрібне більше одного рядка, то на наступних рядках робляться відступи.

В структурі вибору для кожної гілки повинні бути зроблені відступи, щоб показати вміст кожної гілки.

Для виділення операторів, які створюють тіло циклу, складених і

вкладених операторів також використовуються відступи.

В структурі вибору не допускається більше трьох рівнів вкладеності: більшої вкладеності циклів також слід уникати.

Початкові коментарі програми повинні відділятися від операторів порожнім рядком.

Разом з тим слід зазначити, що в різних організаціях, які зайняті розробкою ПЗ, можуть використовуватися свої стандарти кодування, проте принциповою є дисципліна їх дотримання.

7.2.1 *Мета структурного програмування*

Основними цілями структурної методології розробки ПЗ є:

1) Створення високоякісних програм з передбачуваною поведінкою;
2) Створення програм, які просто модифікувати;
3) Спрощення структури програми і процесу їх розробки;
4) Досягнення передбачуваності процесу розробки і покращення управління процесом;
5) Скорочення термінів розробки і зниження вартості розробки ПЗ.

Перераховані цілі доцільно конкретизувати стосовно до методології структурного програмування.

Таблиця 7.1 – Перелік цілей і процесів для їх досягнення

Поліпшити легкість читання програм:	Зробити максимально правильною відповідність між текстом вихідної програми і процесом її виконання; Зменшити складність програми за рахунок спрощення шляхів розгалуження в програмі; Забезпечити можливість читання програми від початку до кінця без „стрибків” в розгалужені програми.
Підвищити ефективність програм:	Зробити програми ефективними з погляду часу виконання основних вимог; Зробити програми ефективними з погляду їх супроводу, тобто структура програми повинна забезпечувати легкість виявлення і виправлення помилок, а також простоту модифікації.

Підвищити надійність програм:	Конструювати програми так, щоб відпадала або зменшувалася необхідність наладки; Конструювати програми так, щоб вони піддавалися повному тестуванню; Використовувати доведення коректності програм, як частину процесу їх конструювання; Ввести більш високий рівень точності в програмуванні.
Створити дисципліну програмування:	Систематизувати процес програмування; Підвищити цілісність програмної системи; Примусити програміста думати.
Зменшити вартість програмування:	Підвищити продуктивність праці програміста; Спростити програмісту управління великими об'ємами кодів програми.

7.2.2 *Емпіричні підходи в програмуванні з використанням покрокової деталізації*

Для розробки структурних програм Вірт запропонував методологію, названу ним програмуванням шляхом покрокової деталізації (вдосконалення).

Покрокова деталізація відрізняється наступними особливостями:

Процес виконується у вигляді послідовності окремих кроків.

На кожному кроці використовується множина задач і структур даних, а програма на кожному кроці описується в термінах цих множин.

Множина задач і даних кожного кроку деталізує множини, отримані на попередньому кроці деталізації.

На кожному кроці використовується свій рівень представлення задач і структур даних, у міру завершення деталізації він наближається до рівня мови програмування.

На кожному кроці рішення про деталізацію приймається після розгляду альтернативних варіантів.

Для опису процесу деталізації Дейкстра ввів поняття рівнів абстракції. Самий верхній рівень опису програми представляє її в найабстрактнішій формі, а самий нижній – в термінах компонентів програмної обстановки, які легко можуть транслювати в програмний код. Кожний рівень абстракції формується з компонентів трьох типів:

Множини структур даних;

Множини інструкцій;

Алгоритм, виражений в термінах даних і інструкцій, створених на даному рівні.

Дейкстра розглядає кожний рівень як машину, що має набір виконуваних інструкцій і обробляє дані певної структури. Спочатку розглядається найабстрактніша (віртуальна) машина, яка оперує, наприклад, з даними на рівні файлів, виконуючи операції типу „прочитати”, „записати”, або „надрукувати” (файл). Потім множина інструкцій і множина даних деталізує і розширяється і наближається до реального комп’ютера.

7.2.3 *Емпіричні компоненти при низхідному і висхідному програмуванні*

Низхідне програмування – систематичний метод проектування, кодування і тестування програм у вигляді послідовності кроків. Метод базується на ідеї покрокової деталізації і дозволяє створити ієрархічно організовану модульну програму, модуль верхнього рівня якої відповідає загальній функції програми, а модулі наступного рівня представляють її підфункції.

Традиційно програма розробляється у вигляді послідовності кроків: спочатку програма проектується, потім вона кодується і, нарешті, тестується. У випадку низхідного програмування спочатку проектується, кодується і тестується один модуль, потім інший і т.д., поки всі модулі не будуть завершені. Програма росте поступово у міру підключення до неї нових модулів. Низхідна розробка програми починається після створення загальної логічної структури програми. Перш за все проектується модуль самого верхнього рівня. Модулі наступного рівня розглядаються як фіктивні і замінюються так званими „заглушками” – порожніми модулями, що містять тільки точки входу і виходу. Такі модулі при виклику просто повертають управління модулю, який їх викликав. Надалі вони замінюються реальними модулями.

Метод висхідного програмування використовує протилежну стратегію. Програма створюється шляхом об’єднання простих програмних компонентів для створення компонент більш високого рівня. Проте першим кроком розробки, як і в попередньому підході, є створення загальної структурної схеми

програми, включаючи функціональні компоненти і структури даних.
Після цього використовується покроковий процес розробки і об'єднання компонентів, починаючи з компонент самого нижнього рівня. При об'єднанні і сумісному тестуванні модулів нижнього рівня виникає необхідність в модулі більш високого рівня, який в даний момент ще не існує. Для імітації його функцій створюється тестовий драйвер – скелетна модель модуля, який буде створений на наступному кроці. Драйвер розглядається як управляюча програма, що викликає інші модулі і передає їм управління.

Обидва методи впорядковують процес програмування. На практиці, проте, більш ефективним вважається комбінований підхід. Так, спільні модулі, які викликатимуться більш ніж однією компонентою більш високого рівня, доцільно розробляти, використовуючи висхідне програмування, а для решти частини програми використовувати низхідний принцип. Висхідний підхід дозволяє працювати паралельно над різними модулями програми, а низхідний спрощує інтеграція модулів в єдину програму.

7.2.4 *Використання емпіричного підходу при модульності ПЗ*

Модуль – це фрагмент тексту програми, який є будівельним блоком для фізичної структури системи. Модулем може бути яка-небудь процедура або функція Як правило, модуль складається з двох частин інтерфейсу і реалізації.

Модульність – це властивість системи, яку можна поділити на ряд модулів, які внутрішньо зв'язані і слабо залежать один від одного. За визначенням Г. Майєрса, модульність – властивість ПЗ, яка забезпечує інтелектуальну можливість створення будь-якої складної програми.

Нехай $C(x)$ – функція складності вирішення проблеми x, $T(x)$ – функція затрат часу на вирішення проблеми x. Для двох проблем $p1$ і $p2$ з співвідношення $C(p1) > C(p2)$ слідує, що $T(p1) > T(p2)$, – вирішення складної проблеми потребує більше часу. Аналогічно можна показати, що $C(p1 + p2) > C(p1) + C(p2)$ і, відповідно $T(p1 + p2) > T(p1) + T(p2)$.

Це співвідношення є обґрунтуванням модульності. Воно приводить до висновку „поділяй і володарюй" – складну проблему легше вирішити,

розділивши її на керовані частини. Проте тут відображено лише частина реальності – не враховуються затрати на інтерфейс між модулями. З збільшенням кількості модулів (і зменшенням їх розміру) ці затрати також ростуть.

Таким чином, існує оптимальна кількість модулів, яка приводить до мінімальної вартості розробки. Проте не має відповідного критерію для передбачення такої кількості. Проте, розробники знають, що оптимальний модуль повинен задовольняти два критерії:

Ззовні він простіший, ніж всередині;

Його простіше використовувати, ніж побудувати.

Модуль повинен мати наступні властивості:

Кожний модуль представляє одну логічну задачу.

Модуль замкнутий і простій.

Модуль дискретний і оглядовий.

Модуль окремо тестується і наладжується.

Кожний модуль реалізується стосовно однієї незалежної функції програми.

Кожний модуль має одну точку входу і одну точку виходу.

Модулі можуть об’єднуватися в більш крупні модулі без знання особливостей їх внутрішнього вмісту.

Модулі повинні мати добре визначені інтерфейси, при цьому управляючі зв’язки здійснюються через їх точки входу і виходу.

При побудові модульної програми важливим представляється вибір розміру модуля. Як груба вказівка використовується розмір модуля в рядках коду або в кількості операторів. В різних джерелах приводяться різні оцінки, узагальнюючи їх, можна сказати, що модуль не повинен містити більше 50 рядків коду.

Міра складності модуля також відіграє важливу роль. Існує декілька підходів до кількісної оцінки складності. Одна з найзручніших і поширених – оцінка за допомогою так званого цикломатичного числа, яке рівне кількості шляхів виконання програми. Таким чином, складність програми або модуля

рівна числу операторів IF + 1. Якщо в умовному операторі використовується складний логічний вираз, то підраховується кількість простих операцій порівняння.

Структура взаємозв'язків модулів програми називається схемою модуляризації. Головна мета модуляризації – отримання схеми, яка повинна бути адекватною складовим частинам вирішуваної проблеми. Якість схеми модуляризації оцінюється простотою і ефективністю її реалізації. Розробка схеми модуляризації – процес евристичний, що дозволяє отримати декілька варіантів рішення, серед яких повинен бути вибраний якнайкращий. Як і всяка ієрархічна структура, схема модуляризації характеризується глибиною – кількістю рівнів ієрархії, і широтою – кількістю модулів на певному рівні. Важливою характеристикою, яка визначає складність програм і якість схеми модуляризації, є поняття віяла управління, яке рівне кількості дочірніх модулів, що викликаються батьківським модулем. Дуже високі або низькі значення віяла управління в схемі модуляризації служать показниками поганої схеми модуляризації. В загальному випадку віяло управління не повинне перевищувати 10, а як близьке до оптимального на практиці розглядають число 7. Доцільно будувати схему модуляризації, в якій віяло управління має невеликі відхилення від цього числа для всіх модулів.

7.2.5 *Інформаційна закритість*

Принцип інформаційної закритості (Д.Парнас, 1972) стверджує: вміст модулів повинен бути прихований один від одного. Модуль повинен визначатися і проектуватися так, щоб його вміст (функції і дані) були недоступні тим модулям, яким не потрібна така інформація.

Інформаційна закритість означає наступне:

1. Всі модулі незалежні, обмінюються лише інформацією, яка необхідна для роботи;

2. Доступ до операцій і структур даних модуля обмежений.

Принцип інформаційної закритості забезпечує можливість розробки модулів різними розробниками та забезпечується легка модифікація системи. Ймовірність розповсюдження помилок дуже мала, так як більшість даних і

процедур приховано від інших частин системи.

Ідеальний модуль відіграє роль „чорного ящика”, вміст якого невидимий клієнтам. Він простий у використанні – кількість „ручок і органів управління” ним невелика. Його легко розвивати і коректувати в процесі супроводу ПЗ. Для забезпечення таких можливостей система внутрішніх і зовнішніх зв’язків модуля повинна відповідати особливим вимогам.

7.3 Зв’язність модуля

Зв'язність, або міцність (англ. Cohesion) - спосіб і ступінь, в якій завдання, що виконуються деякими програмний модулем, пов'язані один з одним; міра сили взаємопов'язаності елементів усередині модуля [1].

Зв'язність зазвичай протиставляється зачеплення.

типи зв'язності

У стандарті ISO / IEC / IEEE 24765 [1] і сучасній літературі [2] [3] [4] пропонується розглядати наступні типи зв'язності:

• випадкова (coincidental);

• комунікаційна (communicational);

• функціональна (functional);

• логічна (logical);

• процедурна (procedural);

• послідовних (sequential) і

• тимчасова (temporal).

Ці види зв'язності аналогічні використовуваним семи видах зв'язності в SADT [5].

Зв’язність модуля – це міра залежності його частин, внутрішня характеристика модуля. Чим вища зв’язність модуля, тим кращий результат проектування, тобто чим „чорніший” його ящик, тим менше „ручок управління” на ньому знаходиться і тим простіші ці „ручки”.

Для вимірювання зв’язності використовують поняття сили зв’язності. Існує 7 типів зв’язності:

1. Функціональна зв’язність (10). Частини модуля разом реалізують одну функцію. Функціонально зв’язаний модуль містить елементи, які беруть участь у виконанні однієї і тільки однієї проблемної задачі. Коли клієнт викликає модуль, виконується тільки одна робота, без залучення зовнішніх обробників. Деякі з функціонально зв’язних модулів дуже прості, інші складні. Не

дивлячись на складність модуля і на те, що його обов'язки виконують декілька функцій, якщо його дії можна представити як єдину проблемну функцію (з точки зору клієнта), тоді вважають, що модуль функціонально зв'язний. Програми, побудовані з функціонально зв'язних модулів, легше супроводжувати. Помилково вважати, що будь-який модуль можна розглядати як однофункціональний – існує багато різновидностей модулів, які виконують для клієнтів перелік різних робіт, і цей перелік не можна розглядати як єдину проблемну функцію. Критерій при визначенні рівня зв'язності цих нефункціональних модулів – як зв'язані один з одним різні дії, які вони виконують.

2. Інформаційна (послідовна) зв'язність (9). Вихідні дані однієї частини використовуються як вхідні дані в іншій частині модуля. Тобто при такій зв'язності елементи-обробники модуля утворюють конвеєр для обробки даних. Супроводжувати модулі з інформаційною зв'язністю легко, але можливості повторного використання модулів невисока.

3. Комунікативна зв'язність (7). Частини модуля зв'язані за даним (працюють з однією і тою ж структурою даних). З точки зору клієнта проблема використання комунікативно зв'язного модуля полягає в надлишковості отримуваних результатів. Майже завжди розбиття комунікативно зв'язного модуля на окремі функціонально зв'язні модулі покращує супроводжуваність системи.

4. Процедурна зв'язність (5). Частини модуля зв'язані порядком виконуваних ними дій, які реалізують деякий сценарій поведінки. При досягненні процедурної зв'язності попадають в пограничну ділянку між доброю (для модулів з більш високими рівнями зв'язності) і поганою супроводжуваністю (для модулів з більш низькими рівнями зв'язності). Процедурно зв'язний модуль складається з елементів, які реалізують незалежні дії, для яких задано порядок роботи, тобто порядок передачі управління. Залежності за даними між елементами немає.

5. Часова зв'язність (3). Частини модуля не зв'язані, але необхідні в один і той же період роботи системи. Недолік: сильний взаємний зв'язок з іншими модулями та велика чутливість до внесення змін. При часовій зв'язності елементи-обробники модуля прив'язані до конкретного періоду часу. Класичним прикладом модуля такого типу зв'язності є модуль ініціалізації.

6. Логічна зв'язність (1). Частини модуля об'єднані за принципом

функціональної подібності. Наприклад, модуль складається з різних підпрограм обробки помилок. При використанні такого модуля клієнт вибирає тільки одну з підпрограм. Недоліками таких модулів є складність спряження та велика ймовірність внесення помилок при зміні спряження заради однієї з функцій.

7. Зв'язність за співпадінням (0). В модулі відсутні явно виражені внутрішні зв'язки.

Зв'язність за співпадінням, логічна та часова зв'язності – результат неправильного планування архітектури, а процедурна зв'язність – результат поганого планування архітектури ПЗ, загальна характеристика типів яких приведена в табл. 7.2.

Таблиця 7.2- Загальна характеристика типів зв'язності

Тип зв'язності	Супроводжуємість	Роль модуля
Функціональна	Кращий супровід	„Чорний ящик"
Інформаційна (послідовна)		Не зовсім „чорний ящик"
Комунікативна		„Сірий ящик"
Процедурна	Поганий супровід	„Білий" або „прозорий ящик"
Часова		„Білий ящик"
Логічна		
За співпадінням		

Алгоритм визначення рівня зв'язності модуля доволі простий.

1. Якщо модуль – це одинична проблемно-орієнтована функція, то рівень зв'язності – функціональний; кінець алгоритму. В іншому випадку перейти до 2.

2. Якщо дії в модулі зв'язані, то перейти до 3. Якщо дії ніяк не зв'язані, то перейти до 6.

3. Якщо дій в модулі зв'язані даними, то перейти до 4, якщо потоком управління – до 5.

4. Якщо порядок дій в модулі важливий, то рівень зв'язності – інформаційний, інакше – комунікативний. Кінець алгоритму.

5. Якщо порядок дій в модулі важливий, то рівень зв'язності – процедурний, інакше часовий. Кінець алгоритму.

6. Якщо дії в модулі належать до однієї категорії, то рівень зв'язності – логічний, інакше –за співпадінням. Кінець алгоритму.

Можливі більш складні випадки, коли з модулем асоціюються декілька рівнів зв’язності. В цих випадках потрібно застосовувати одно з двох правил:

Правило паралельності. Якщо всі дії модуля мають декілька рівнів зв’язності, то модулю присвоюють самий сильний рівень.

Правило послідовності. Якщо дії в модулі мають різні рівні зв’язності, то модулю присвоюють самий слабкий рівень.

7.4 Слабка зв'язність. Закон Деметри

Закон Деметри (англ. Law of Demeter, LoD) - набір правил проектування при розробці програмного забезпечення, зокрема об'єктно-орієнтованих програм, який накладає обмеження на взаємодії об'єктів (модулів). Узагальнено, закон Деметри є спеціальним випадком слабкого зачеплення (Loose coupling). Правила були запропоновані в кінці 1987 в північно-східному Університеті (Бостон, Массачусетс, США).

Говорячи спрощено, кожен програмний модуль:

• повинен володіти обмеженим знанням про інші модулях: знати про модулях, які мають «безпосереднє» ставлення до цього модулю.

• повинен взаємодіяти тільки з відомими йому модулями «друзями», що не взаємодіяти з незнайомцями.

• звертатися тільки до безпосередніх «друзям».

Аналогія з життя: Якщо Ви хочете, щоб собака побігла, нерозумно командувати її ногами, краще віддати команду собаці, а вона вже розбереться зі своїми ногами сама.

Основною ідеєю є те, що об'єкт повинен мати якомога менше уявлення про структуру і властивості чого завгодно (включаючи власні подкомпоненти).

Назва взято з проекту «Деметра», який використовував ідеї аспектно-орієнтованого та адаптивного програмування. Проект був названий на честь Деметри, грецької богині землеробства, щоб підкреслити достоїнства філософії програмування «знизу-вгору».

В об'єктно-орієнтованому програмуванні

Загальний опис правила: Об'єкт A не повинен мати можливість отримати безпосередній доступ до об'єкта C, якщо в об'єкта A є доступ до об'єкта B і в об'єкта B є доступ до об'єкта C.

Більш формально, Закон Деметри для функцій вимагає, що метод M

об'єкта Про повинен викликати методи тільки таких типів об'єктів:

• власне самого Про

• параметрів М

• інших об'єктів, створених в рамках М

• прямих компонентних об'єктів Про

• глобальних змінних, доступних О, в межах М

Практично, об'єкт-клієнт повинен уникати викликів методів об'єктів, внутрішніх членів, повернутих методом об'єкта-сервісу.

Для багатьох сучасних об'єктно-орієнтованих мов програмування, що використовують точку, як оператор доступу до членів класу, закон може бути перефразувавши як «Використовуйте тільки одну точку».

Використання процесу впровадження залежностей сприяє [1] дотриманню Закону Деметри.

Багатоярусна архітектура може також розглядатися як приклад реалізації закону Деметри в програмній системі. У такій архітектурі код кожного ярусу може викликати тільки код свого і нижчого ярусу. Виклик «через ярус» є порушенням багатоярусної архітектури.

приклад коду

Таким чином, код abMethod () порушує Закон Деметри, а код a.Method () є коректним.

Переваги та недоліки

Перевагами закону Деметри є те, що код, розроблений з дотриманням цього закону, робить написання тестів більш простим [2], а розроблене програмне забезпечення менш складно за підтримки і має великі можливості повторного використання коду. Так як об'єкти є менш залежними від внутрішньої структури інших об'єктів, контейнери об'єктів можуть бути змінені без модифікації викликають об'єктів (клієнтів).

Недоліком закону Деметри є те, що іноді потрібно створення великої кількості малих методів-адаптерів (делегатів) для передачі викликів методу до внутрішніх компонентів.

7.5 Зчеплення модулів

Зачеплення, або зчеплення (англ. Coupling) - спосіб і ступінь взаємозалежності між програмними модулями [1]; сила взаємозв'язків між модулями [2]; міра того, наскільки пов'язані підпрограми або модулі [1].

Зчеплення звичайно протиставляється зв'язності (англ. Cohesion). Слабке зачеплення часто поєднується з сильною зв'язністю і навпаки. Метрика якості ПЗ пов'язаності і зв'язності була придумана Ларрі Костянтином, початковим розробником структурного проектування [3], який був також раннім прихильником таких концепцій (див. Також SSADM). Слабке зчеплення часто є ознакою добре структурованої комп'ютерної системи і ознакою хорошого проекту, і, коли вона комбінується з сильною зв'язністю, відповідає загальним показникам хорошої читаності і супроводу.

Концептуальна модель зчеплення має вигляд на Рис.7.1.

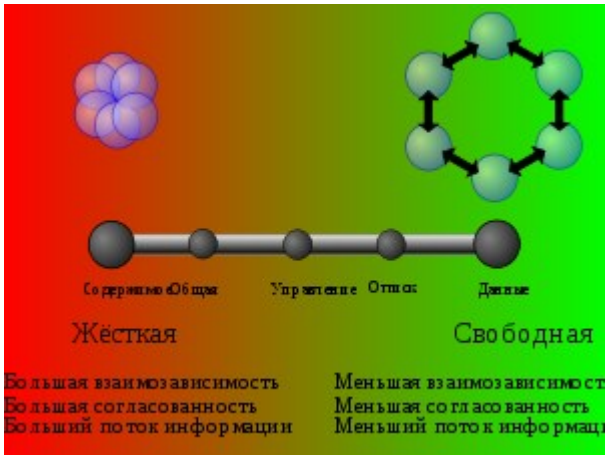


Рисунок 7.1 - Концептуальна модель зчеплення, яка включає два полюси взаємозалежності між програмними модулями

Типи зчеплення, відповідно до стандарту ISO / IEC / IEEE 24765-2010, включають:

• зчеплення за загальною області (common-environment coupling);

• зчеплення по вмісту (content coupling);

• зчеплення з управління (control coupling);

• зчеплення за даними (data coupling);

• змішане зчеплення (hybrid coupling) і

• патологічне зчеплення (pathological coupling) [1].

Зчеплення модулів – це міра взаємозалежності модулів за даними, це зовнішня характеристика модуля, яку бажано зменшувати. Три чинники роблять вплив на зчеплення модулів:

Кількість елементів даних, які передаються між ними.

Кількість управляючих даних, які передаються між модулями.

Кількість глобальних елементів даних, які спільно використовуються модулями.

Перший чинник – це кількість параметрів, які передаються в модуль, що викликається. Часто передача багатьох параметрів показує, що модуль виконує не одну, а декілька функцій. На складність програми робить вплив не тільки кількість, але й тип даних, які передаються. Доцільно використовувати в якості параметрів не структурні, а прості дані. Дані, що спільно використовуються, – дані, які явно згадуються в модулі, а не передаються у вигляді параметрів. Найяскравішим прикладом служать глобальні дані. Використання глобальних даних ускладнює розуміння і особливо зміну окремих модулів, оскільки їх дані можуть використовуватися непомітним чином і різними шляхами.

Кількісно зчеплення вимірюється мірою зчеплення і виділяють 6 типів.

1. Зчеплення за даними (1). Модуль А викликає модуль В. Усі вхідні й вихідні параметри викликаного модуля – прості елементи даних.

2. Зчеплення за зразком (3). В якості параметрів використовуються структури даних.

3. Зчеплення за управлінням (4). Модуль А явно управляє функціонуванням модуля В (за допомогою прапорців або перемикачів), посилаючи йому управляючі дані.

4. Зчеплення за зовнішніми посиланнями (5). Модулі А і В посилаються на один і той же глобальний елемент даних.

5. Зчеплення за спільною пам’яттю (7). Модулі розділяють одну й ту ж глобальну структуру даних.

6. Зчеплення за вмістом (9). Один модуль прямо посилається на вміст іншого модуля (не через його точку входу). Наприклад, коди їх команд перемішуються один з одним.

8 Емпіричні методи тестування програмного забезпечення

8.1. Поняття та принципи тестування

Тестування – це процес виконання програми з метою виявлення помилок. Кроки процесу задаються тестами. Кожен тест визначає свій набір вихідних даних і умов для запуску програми та набір очікуваних результатів роботи програми. Повну перевірку програми гарантує вичерпне тестування. Воно потребує перевірити всі набори вихідних даних, всі варіанти їх обробки і

включає велику кількість тестових варіантів. Проте вичерпне тестування в більшості випадків залишається тільки мрією – спрацьовують ресурсні обмеження. Хорошим вважають тестовий варіант з високою ймовірністю виявлення ще не розкритої помилки. Успішним називають тест, який виявляє до сих пір не розкриту помилку.

Метою проектування тестових варіантів є систематичне виявлення різних класів помилок при мінімальних затратах часу і коштів. Тестування забезпечує:

а) Виявлення помилок;

б) Демонстрацію відповідності функцій програми її призначенню;

в) Демонстрацію реалізації вимог до ПЗ;

г) Відображення надійності як індикатора якості програми.

д) Тестування не може показати відсутність дефектів.

На вході процесу тестування існує три потоки: текст програми, вихідні дані для запуску програми і очікувані результати. В процесі тестування виконуються тести і всі отримані результати оцінюються Це значить, що реальні результати тестів порівнюються з очікуваними результатами Коли виявляється неспівпадіння, фіксується помилка – починається наладка Час процесу наладки неможливо передбачити.

Існує два принципи тестування програм – функціональне тестування (тестування „чорного ящика”) і структурне тестування (тестування „білого ящика”).

8.2 Тестування „чорного ящика”

При такому способі тестування відомі функції програми і досліджується робота кожної функції на всій області визначення. Основне місце застосування тестів „чорного ящика” – інтерфейс ПЗ. Ці тести демонструють:

Як виконуються функції програм;

Як приймаються вихідні дані;

Як виробляються результати;

Як зберігається цілісність зовнішньої інформації.

При тестуванні „чорного ящика” розглядаються системні характеристики програм, ігнорується їх внутрішня логічна структура. Вичерпне тестування, як правило, неможливе. Тестування „чорного ящика” не реагує на більшість особливостей програмних помилок.

Тестування „чорного ящика” (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми. Програмний виріб тут розглядається як „чорний ящик”, чию поведінка можна дослідити тільки дослідженням його входів і відповідних виходів. При такому підході бажано мати:

набір, який утворюють такі вхідні дані, які приводять до аномалій у поведінці програми;

набір, який утворюють такі вихідні дані, які демонструють дефекти програми.

В багатьох випадках визначення таких тестових варіантів базується на попередньому досвіді інженерів тестування.

Принцип „чорного ящика” дозволяє виявити наступні категорії помилок:

1. Некоректні або відсутні функції.

2. Помилки інтерфейсу.

3. Помилки в зовнішніх структурах даних або в доступі до зовнішньої бази даних.

4. Помилки характеристик ПЗ.

5. Помилки ініціалізації і завершення.

Таке тестування застосовують на пізніх стадіях розробки. При тестуванні ігнорують управляючу структуру програми. Увага концентрується на інформаційній галузі.

8.3 Тестування „білого ящика”

При такому тестуванні відома внутрішня структура програми і досліджуються: внутрішні елементи програми і зв’язки між ними. Об’єктом тестування є не зовнішня, а внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми і правильність їх взаємодії. Звичайно аналізуються управляючі зв’язки елементів, рідше – інформаційні зв’язки. Тестування за принципом „білого ящика” характеризується мірою, в якій тести виконують або покривають логіку програми. Вичерпне тестування також неможливе.

Тестування „білого ящика” базується на аналізі управляючих структур програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графу управління.

Недоліками тестування „білого ящика” є:

1. Кількість незалежних маршрутів може виявитися дуже великою.

2. Вичерпне тестування маршрутів не гарантує відповідності програми до вихідних вимог.

3. В програмі можуть бути пропущені деякі маршрути.

4. Не можна виявити помилки, поява яких залежить від оброблюваних даних.

Переваги тестування „білого ящика” зв’язані з тим, що принцип „білого ящика” дозволяє врахувати особливості програмних помилок:

1. Кількість помилок мінімальна в „центрі” і максимальна на „периферії” програми.

2. Попередні припущення про ймовірність потоку управління або даних в програмі часто бувають некоректними. В результаті типовим може стати маршрут, модель обчислень за яким пророблена не достатньо.

3. При запису алгоритму ПЗ у вигляді тексту мовою програмування можливе внесення типових помилок трансляції.

4. Деякі результати в програмі залежать не від вхідних даних, а від внутрішнього стану програми.

Кожна з цих причин є аргументом для проведення тестування за принципом „білого ящика”. Тести „чорного ящика” не можуть реагувати на помилки таких типів.

8.3.1 *Тестування базового шляху*

Тестування базового шляху (Том МакКейб, 1976) – це спосіб, який базується на принципі „білого ящика”. Цей спосіб дає можливість отримати оцінку комплексної складності програми і використовувати її для визначення необхідної кількості тестових варіантів.

Тестові варіанти розробляються для перевірки базової множини шляхів (маршрутів) в програмі. Вони гарантують однократне виконання кожного оператора програми при тестуванні.

Для представлення програми використовується потоковий граф, який має особливості.

1. Граф будується відображенням управляючої структури програми. В ході відображення закриваючі дужки умовних операторів і операторів циклів розглядаються як окремі оператори.

2. Вузли потокового графу відповідають лінійним ділянкам програми, включають один або декілька операторів програми.

3. Дуги потокового графу відображають потік управління в програмі.

4. Розрізняють операторні і предикатні вузли. З операторного вузла виходить одна дуга, а з предикатного – дві.

5. Предикатні вузли відповідають простим умовам в програмі. Складені умови програми відображаються в декілька предикатних вузлів.

6. Замкнуті ділянки, які утворені дугами і вузлами, називають регіонами.

7. Середовище, яке оточує граф розглядається як додатковий регіон.

Цикломатична складність – метрика ПЗ, яка забезпечує кількісну оцінку логічної складності програми. В способі тестування базового шляху цикломатична складність визначає кількість незалежних шляхів в базовій множині програми і верхню оцінку кількості тестів, які гарантують однократне виконання всіх операторів. Незалежним називається будь-який шлях, який вводить новий оператор обробки або нову умову. В термінах потокового графу незалежний шлях повинен містити дугу, яка не входить в раніше визначений шлях. Усі незалежні шляхи графа утворюють базову множину, яка має наступні властивості:

тести, які забезпечують його перевірку, гарантують однократне виконання кожного оператора і виконання кожної умови;

потужність базової множини рівна цикломатичній складності потокового графу.

Цикломатична складність обчислюється одним із трьох способів:

1. Цикломатична складність рівна кількості регіонів потокового графу.

2. Цикломатична складність визначається за формулою $V(G) = E - N + 2$, де E – кількість дуг, N – кількість вузлів потокового графу.

3. Цикломатична складність формується за виразом $V(G) = p + 1$, де p – кількість предикатних вузлів в потоковому графі G.

8.3.2 *Способи тестування умов*

Мета цих способів тестування – будувати тестові варіанти для перевірки логічних умов програми. При цьому бажано забезпечити залучення операторів із всіх гілок програми.

Елементами умови є: логічний оператор, логічна змінна, пара дужок, яка містить просту або складену умову, оператор відношення, арифметичний вираз.

Ці елементи визначають типи помилок в умовах.

Якщо умова некоректна, то некоректний хоч один із елементів умови –в умові можливі наступні типи помилок:

1) Помилка логічного оператора (наявність некоректних/відсутніх/ надлишкових логічних операторів).
2) Помилка логічної змінної.
3) Помилка логічної дужки.
4) Помилка оператора відношення.
5) Помилка арифметичного виразу.

Спосіб тестування умов орієнтований на тестування кожної умови в програмі. Методики тестування умов мають дві переваги. По-перше, достатньо просто виконати вимірювання тестового покриття умови. По-друге, тестове покриття умов в програмі – це фундамент для генерації додаткових тестів програми.

Ціллю тестування умов є визначення не тільки помилок в умовах, але й інших помилок в програмах. Існує декілька методик тестування умов.

Найпростіша методика – тестування гілок. В ній для складеної умови перевіряється: кожна проста умова, яка входить в неї, гілка True, гілка False.

Інша методика – тестування ділянки визначення. В ній для виразу відношення потрібна генерація 3-4 тестів.

8.3.3 *Тестування циклів*

Цикл – найбільш розповсюджена конструкція алгоритмів. Тестування циклів проводиться за принципом „білого ящика”, при перевірці циклів основна увага приділяється правильності конструкцій циклів. Розрізняють 4 типи циклів: прості, вкладені, об’єднані, неструктуровані.

Для перевірки простих циклів з кількістю повторень n може використовуватися один з наступних наборів тестів:

1. Прогін всього циклу.
2. Тільки один прохід циклу.
3. Два проходи циклу.
4. m проходів циклу, де $m < n$.
5. $n-1, n, n+1$ проходів циклу.

Із збільшенням рівня вкладеності циклів кількість можливих шляхів різко

зростає. Це приводить до нереальної кількості тестів. Для скорочення кількості тестів застосовується спеціальна методика, в якій використовуються такі поняття, як оточуючий і вкладений цикли.

Для тестування за цією методикою передбачено наступні кроки:

1. Вибирається самий внутрішній цикл. Встановлюються мінімальні значення параметрів всіх інших циклів.

2. Для внутрішнього циклу проводяться тести простого циклу. Добавляються тести для виключених значень і значень, які виходять за межі робочого діапазону.

3. Переходять в наступний оточуючий цикл. Виконують його тестування. При цьому зберігаються мінімальні значення параметрів для всіх зовнішніх циклів і типові значення для всіх вкладених циклів.

4. Робота продовжується до тих пір, поки не будуть відтестовані всі цикли.

Якщо кожен із циклів незалежний від інших, то використовується техніка тестування простих циклів. При наявності залежності використовується методика для вкладених циклів.

Неструктуровані цикли неможливо тестувати. Цей тип циклів потрібно переробити за допомогою структурованих програмних інструкцій.

8.4 Налаштування програмного забезпечення

Налаштування (наладка) – це процес знаходження місцеположення помилок в програмі і їх виправлення. Процес наладки починається після виявлення факту помилки в результаті тестування і здійснюється в два етапи.

Перший етап – встановлення причини помилки і її локалізація (визначення її місцеположення в програмі); другий – виправлення помилки і перевірка правильності роботи програми.

Таким чином, основним засобом виявлення помилок при наладці є тестування. Тестування і наладка при цьому звичайно проводяться одночасно. Прийнято виділяти три стадії тестування:

1. Для виявлення помилки в програмі;
2. Для встановлення місцезнаходження помилки;
3. Для підтвердження правильності роботи програми після проведеного коректування.

Як показує досвід розробки ПЗ, трудомісткість наладки перевищує

сумарну трудомісткість розробки алгоритму, програмування (кодування) і тестування. Витрати часу на наладку складають від 50 до 80% загального часу розробки програми, тому наладку іноді називають мистецтвом виявлення місцеположення помилок в програмі.

В результаті розробки численних програмних виробів було встановлено, що навіть досвідчені програмісти зазнають серйозні труднощі у виявленні причини помилки. Вони не знають, які з помилок найбільш вірогідні, які ділянки програм схильні до помилок більшою мірою, які стратегії пошуку помилок найбільш ефективні і т.п. Це обумовлено тим, що звичайно навчають алгоритмізації і програмуванню задач, але не вивчають наладку й тестування. Окрім цього, програмісти відносяться до процесу наладки, як правило, негативно, віддаючи перевагу більш творчому процесу розробки алгоритму.

До причин, що визначають значну трудомісткість процесу наладки, слід віднести і порушення програмістами дисципліни структурної методології, тобто принципу формальності, який вимагає дотримуватися при проектуванні і кодуванні формальних правил і методів розробки структурних програм. Виконання структурних вимог дозволяє уникнути більшості помилок і помітно спростити процедуру наладки програми. На жаль, багато програмістів прагнуть найшвидше написати програму, а потім знаходити помилки шляхом багатократного її виконання з різноманітними тестовими даними без їх уважного аналізу і ретельного проектування. Такий підхід приводить до значних витрат часу при встановленні причин помилок і їх локалізації, а також до помітного зниження надійності програм, що надалі виявляється на етапі супроводу програмного виробу.

Машинні методи наладки для досвідчених і творчих програмістів виявляються менш ефективними. Як показують дослідження, найістотнішим для підвищення ефективності наладки є апріорні знання про статистику помилок і їх вірогідні типи, а також знання про структуру програми і про ділянки програми, найбільшою мірою схильні до помилок. Такими ділянками, як показує досвід, є перш за все ділянки з високою складністю.

З погляду організації робіт по наладці програми найефективнішим вважається груповий метод відшукання помилок, коли два програмісти спочатку незалежно, а потім спільно здійснюють цей процес. Такий підхід більш досконалий, ніж індивідуальна робота одного програміста. Підвищення ефективності роботи по виявленню і локалізації помилок може бути досягнута

шляхом змінного уважного аналізу програми за столом і машинного тестування.

До труднощів наладки програм слід також віднести і різноманітність ситуацій, що виникають перед початком її виконання. Ситуації можуть бути наступними:

а) Компілятор не видає повідомлень про помилки, але програма не компілюється.

б) Програма відкомпілювалася, але при виконанні не видає ніяких результатів.

в) Програма відкомпілювалася, але при виконанні відбувається передчасна зупинка.

г) Програма зациклюється.

д) Програма видає невірні результати.

8.5 Засоби і методи виявлення помилок в ПЗ та його наладки

Наладка починається після виявлення помилки. Існують наступні засоби виявлення причини помилки і її локалізації:

1) Текст вихідного коду (лістінг).
2) Докладна специфікація програми.
3) Детальний алгоритм програми.
4) Вихідний лістинг.
5) Аналіз послідовності виконання операторів і оцінка очікуваних значень змінних.
6) Відстеження звернень до підпрограм.
7) Дампи пам’яті.

Ці засоби допускають проведення уважного аналізу програми за столом.

Істотну допомогу в наладці надають інструментальні засоби:

а) Налагоджувальні компілятори відповідної мови програмування;

б) Спеціальні засоби розширення мови програмування для контролю типів і діапазонів значень даних, обробки виняткових ситуацій і т.д.;

в) Засоби для друку значень змінних, що використовуються, при аварійному завершенні програми, для трасування значень змінних в процесі виконання програми і т.п.;

г) Пакети програм для дослідження потоків управління і даних в програмі, контролю індексів і реєстрації викликів підпрограм;

д) Генератори тестових даних, які формують тестові набори даних відповідно до специфікацій, що задаються користувачем;

е) Спеціальні он-лайнові наладчики, що забезпечують автоматизацію рестартів, зупинок і переривань програми, переглядання роботи окремих операторів і т.п.;

ж) Пакети словників/довідників даних, що дозволяють контролювати імена і типи даних і їх використання різними модулями програми;

з) CASE-засоби для побудови схем потоків даних, моделей даних, схем алгоритмів і т.п.;

и) Автоматизовані робочі місця програмістів, які включають перераховані засоби.

Для наладки застосовується ряд методів, які можна укрупнено представити у вигляді груп:

Метод „грубої сили”. Найпоширеніший підхід і традиційно використовується програмістами; пов’язаний зі всебічним аналізом за столом вихідного коду і алгоритму програми, вихідних результатів і повідомлень компілятора. Раніше широко використовувався аналіз вмісту пам’яті (дампу пам’яті) звичайно в шістнадцятковій або вісімковій формі.

Для підвищення ефективності наладки в текст програми включають оператори налагоджувального коду. Найбільш просто здійснити вставку операторів, які реєструють результати виконання конкретного оператора. Часто – це друк вибіркових значень змінних. Крім того, може здійснюватися перевірка завершення логічних ділянок програм, результати виконання умов і т.п. Такий метод вставок передбачає, що після операторів, що перевіряються, включаються оператори, які реєструють результати їх виконання. Після завершення наладки програми налагоджувальні оператори можна залишити у вигляді коментарів для можливого використання їх надалі на етапі супроводу програмного виробу.

На жаль, подібний підхід вимагає використання великого об’єму тестових даних, оскільки процедура локалізації помилок достатньо випадкова. Хоча програмісту надається можливість працювати за терміналом, аналізуючи роботу програми в динаміці і використовуючи при цьому різноманітні інструментальні налагоджувальні засоби, ефективність його роботи методом проб і помилок залишається низькою.

Досвід розробки особливо складних програм показує, що більш

раціонально шукати місцеположення помилки не шляхом багатократного виконання програми з випадковими тестовими наборами даних, а шляхом систематичного і ретельного обдумування і аналізу вирішуваної задачі.

Метод індукції. Велика частка помилок може бути локалізованою в результаті аналізу алгоритму вирішуваної задачі, використовуючи стратегію руху від часткового до загального. В результаті тестування розробник одержує дані, які відображають як правильні, так і неправильні дії програми. Дані повинні бути систематизовані і добре структуровані, з вказівкою симптомів помилки, місця і часу її появи. Одночасно вказуються тестові набори даних, що приводять до невірних результатів, і ті, які дають правильний результат. В результаті аналізу цих даних і взаємозв'язків між різними ознаками помилки виявляються певні закономірності і формулюється гіпотеза про причини помилки.

Для доведення правильності висунутої гіпотези необхідно показати, що вона повністю пояснює всі знайдені симптоми помилки. В протилежному випадку гіпотеза відкидається і процес необхідно повторити, зібравши заздалегідь додаткові дані для висунення нової гіпотези.

Метод дедукції. Метод передбачає, що на основі результатів тестування висувається множина можливих гіпотез про причину помилки. Потім із загального списку виключаються припущення, які суперечать даним тестування. Якщо в результаті аналізу будуть виключені всі висунуті гіпотези, то необхідно за допомогою тестування зібрати додаткові дані і повторити процедуру висунення нових припущень про причину помилки. Коли залишається декілька припущень, всі вони детально аналізуються, починаючи з найправдоподібнішого. Вибрана гіпотеза всебічно розглядається і уточнюється. Доведення її правильності здійснюється, як в попередньому методі, і, якщо вона виявляється вірною, на її основі знаходиться помилка.

Інверсне дослідження логіки програми. Для невеликих програм аналіз логіки виконання програми у зворотному напрямі виявляється досить ефективним способом виявлення помилки. Наладка починається з точки програми, де був знайдений невірний результат або відбулася зупинка програми. На основі отриманих в цій точці значень змінних необхідно визначити, виходячи з логіки програми, які результати повинні були бути при правильній роботі програми. Послідовне просування до початку програми

дозволяє достатньо швидко і точно визначити місце (і причину виникнення) помилки, тобто місце між оператором, де результат виконання програми відповідав очікуваному, і оператором, в якому з'явилися розбіжності.

8.6. Категорії помилок в програмному забезпеченні

Різноманіття помилок, що зустрічаються в програмних виробах, породжує множину їх класифікацій, необхідних для систематизації великого статистичного матеріалу, накопиченого при створенні і експлуатації програмного забезпечення.

Помилки можна систематизувати за етапами розробки програмного продукту.

При описі вимог до програмного продукту появлятися помилки в специфікаціях, обумовлені неповним або некоректним розумінням вимог користувача розробником. неякісно сформульовані користувачем вимоги також породжують помилки у вимогах до програмного виробу. Це найсерйозніші помилки, виправлення яких на подальших етапах проектування, і особливо в процесі експлуатації, може потребувати величезні трудовитрати.

На етапі системного аналізу вирішуваної проблеми можуть також з'являтися неточні формулювання вимог до програмного забезпечення, а також пропуск окремих вимог.

Архітектурне проектування програмного виробу може породжувати помилки, пов'язані з неточностями у функціональній декомпозиції і в інтерфейсах між модулями.

При алгоритмізації окремих функціональних задач можуть бути допущені помилки, обумовлені вибором невірного або неефективного алгоритму, неправильного математичного методу вирішення, невірною логікою алгоритму.

Серйозні помилки можуть бути допущені при проектуванні бази даних (в описі моделі, визначенні типів елементів і структур даних, а також зв'язків між ними, способах звернень до даних і т.п.).

Найбільша кількість помилок відноситься до етапу кодування і тестування. Помилки цієї групи були досліджені якнайповніші, і для них зібраний широкий статистичний матеріал.

Помилки при програмуванні можна просто розділити на синтаксичні і логічні. Багато синтаксичних помилок знаходить транслятор, але й немало таких помилок при трансляції не виявляється. Так у ряді мов програмування не

виявляються пропуски операторів в програмі, порушення форматів вводу-виводу, правил індексації масивів, відсутність початкових значень змінних і т.п. Не виявляються, як правило, і помилки у вказівці типів переданих параметрів при викликах процедур.

Найбільш поширені наступні помилки, які показані в Табл.8.1 зафіксовані при розробці програмного забезпечення.

Таблиця 8.1 - Види і типи помилок при проектуванні ПЗ

Помилки в специфікаціях	Неповна або неоднозначна специфікація Некоректне визначення проблеми
Помилки проектування	Нерозуміння специфікацій Некоректний алгоритм вирішення задачі Пропуск окремих кроків і варіантів алгоритму Помилки ініціалізації змінних Помилки в управлінні циклом Пропуски окремих типів даних
Помилки кодування	Неправильне розуміння проектних документів Помилки в управляючих структурах Помилки форматів вводу-виводу Помилки індексації Помилки ініціалізації і повторна ініціалізація Суперечливі імена змінних Помилки міжпрограмних інтерфейсів Помилки в записі математичних виразів Переповнення, втрата значущості або точність Логічні помилки Відсутність перевірки прапорця і контролю встановлення початкових значень Помилки в операціях маніпулювання даними Помилки в інтерфейсі користувача і в поєднанні з системним ПЗ
Помилки тестування і наладки	Неадекватні тестові набори даних Недостатнє або некоректне використання тестових варіантів і даних Неправильна інтерпретація результатів тестування Неправильні висновки про причини помилки і їх джерела Неправильне розуміння програмної специфікації при виборі тестових наборів даних
Помилки в описі бази даних	Помилки в об’єднанні з базою даних Помилки в словниковій базі метаданих Помилки в ініціалізації бази даних
Канцелярські помилки	Описки Невірне використання клавіш Пропуск або перестановка рядків програми

Зовнішні помилки	Відмови технічних пристроїв Реакція програмного забезпечення на збої в роботі технічних пристроїв Помилки через аварійні ситуації, що виникають в інших системах, з якими взаємодіє даний програмний виріб Помилки в документації

Вірогідність появи помилок перерахованих категорій залежить від багатьох чинників, але, як показує аналіз, найбільш поширені логічні помилки (20—35%), помилки маніпулювання даними, помилки зовнішніх і внутрішніх інтерфейсів, описів даних (кожна категорія від 5 до 20%).

9 Застосування Емпіричних методів на етапах експлуатації та супроводу програмного виробу

9.1. Етап передачі програмного виробу в експлуатацію

Передача в експлуатацію – наступна фаза ЖЦПЗ. Її мета – встановити виріб в робочих умовах і продемонструвати користувачу, що програмний виріб реалізує всі можливості, які були описані в документі „Вимоги користувача”. За установку і випробування програмного виробу несе відповідальність розробник. Основним документом цієї фази служить документ про передачу програмного виробу замовнику, який документально фіксує результати діяльності, пов’язаної з приймальними випробуваннями.

Роботи цієї фази проводяться відповідно до планів, визначених під час попередньої фази, а вхідними матеріалами для передачі в експлуатацію є „Документ детального проектування”, програми і вимоги до приймального тестування.

Перший захід, виконуваний в цій фазі, – контроль всіх представлених матеріалів і установка програмного виробу для виконання проектних функцій. Процедура установки або створення програмного виробу з компонентів може розрізнятися залежно від типу програмного виробу. Установкою програмного виробу звичайно займається штат відділу супроводу, який надалі повинен буде розробляти процедури модифікації. Найважливішою задачею цієї фази є тимчасове (попередня) приймання виробу. Необхідні для її проведення приймальні тести, процедури і план приймального тестування визначаються в плані верифікації і тестування програмного виробу. Тести виконуються під час фази передачі в експлуатацію, вони атестують програмний виріб, тобто демонструють його можливості в робочих умовах. Критерієм для приймання

виробу служить його готовність для використовування. Період робочої перевірки звичайно такий, щоб показати, що програмний виріб відповідає всім вимогам користувача, представленим у відповідному документі.

Рішення про попереднє приймання програмного виробу може бути зроблено користувачем (групою користувачів) після консультацій з операційним персоналом. Рішення про попереднє (тимчасове) приймання повинне бути затверджено і передано розробнику. Попереднє приймання означає закінчення фази передачі програмного виробу в експлуатацію.

Мета документа про передачу – ідентифікувати програмний виріб, який був переданий в експлуатацію, і описати, як він був побудований і встановлений. Документ повинен містити зведений звіт про приймальні випробування і всю документацію про зміни, внесені під час фази приймання.

9.2.Етап планування випробувань

План випробувань – документ, який забезпечує програміста докладним описом всіх тестів з їх розподілом по функціях програмного виробу. Документ містить:

а) Загальні відомості;

б) План випробувань;

в) Технічні вимоги і оцінку результатів;

г) Опис випробувань.

У плані випробувань описується програмний виріб, що підлягає випробуванню, і перераховуються вимоги до необхідних програмних і технічних ресурсів, приводиться графік проведення випробувань. Окрім цього, тут вказуються матеріали, необхідні для проведення випробувань (документація, випробовувані програмні засоби на відповідних носіях, контрольні приклади з очікуваними вихідними результатами).

Особливу роль відіграє розділ документа, в якому детально описується розподіл тестів по функціях програмного виробу; характеризується кожний тест і описується вся послідовність виконання тестів для здійснення всього комплексу перевірок; визначаються загальна методологія і умови проведення випробувань, включаючи тип використовуваних тестових даних, параметри потоків тестових даних і повнота випробувань (повні, часткові, вибіркові).

При описі процедур перевірки програмного виробу обов’язково фіксуються обмеження, пов’язані з умовами випробувань (технічні засоби,

програмне середовище, заповнене бази даних).

В завершенні цього розділу описуються правила оцінки результатів випробувань з врахуванням використаних комбінацій вхідних даних, часу випробувань, обсягів перевірок, а також методи маніпулювання вхідними даними, які повинні бути представлені в найбільш зручній для сприйняття формі.

Останній розділ документа детально описує кожний тест забезпечений ідентифікатором, описується спосіб організації його прогону на комп'ютері і характеризується спосіб управління тестуванням. Тут також описуються вхідні дані і команди вводу тесту, вихідні результати, очікувані при прогоні тесту і можливі проміжні повідомлення.

В документі передбачається методика реєстрації результатів випробувань і супутньої інформації.

ПЕРЕЛІК ПОСИЛАНЬ

До розділів:

Теорія ймовірності

1. Гмурман В. Е. Теория вероятностей и математическая статистика : учеб. пособие для бакалавров / В. Е. Гмурман. — 12-е изд. — М.: Издательство Юрайт, 2013. — 479 с. : ил. — Серия : Бакалавр. Базовый курс. [стр. 17–26].
2. Чжун К.Л., АитСахлиа Ф. Элементарный курс теории вероятностей. Стохастические процессы и финансовая математика. — Пер. с англ. — М.: БИНОМ. Лаборатория знаний., 2007. — 455 с. [стр. 31–53].
3. Прохоров Ю.В., Розанов Ю.А. Теория вероятностей (Основные понятия. Предельные теоремы. Случайные процессы). — М.: Наука, 1973. — 494 с. [стр. 39–45].

Статистичний аналіз коду

1. Скринкаст: Статический анализ Си++ кода // Блог компании PVS-Studio, Habrahabr
2. О безошибочных программах // «Открытые системы», № 07, 2004
3. Первые шаги к решению проблемы верификации программ // «Открытые системы», № 08, 2006
4. Сертификация и тестирование программного обеспечения // НПО Эшелон
5. Что такое «Parallel Lint»? // Viva64
6. Статический анализ безопасности кода // Программная инженерия и информационная безопасность. 2013 № 1, стр 50

7. Лагутин М.Б. Наглядная математическая статистика: учебное пособие — М.: БИНОМ. Лаборатория знаний., 2009. — 472 с. [стр. 71–76].
8. Кривенцов А.С., Ульянов М.В. Интервальная оценка параметров бета-распределения при определении доверительной трудоемкости алгоритмов // Известия ЮФУ. 2012. №7(132). С. 210–219.

Зв'язність

1. ISO/IEC/IEEE 24765-2010 Systems and software engineering — Vocabulary
2. Бадд, 1997, 17.1.2. Разновидности связности
3. Вендров А. М. CASE-технологии. Современные методы и средства проектирования информационных систем. 2.2.3. Типы связей между функциями
4. Пирогов В. Ю. Информационные системы и базы данных: организация и проектирование — СПб, БХВ-Петербург, 2009. С.203-204
5. ISO/IEC TR 19759:2005, Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK)
6. W. Stevens, G. Myers, L. Constantine, «Structured Design», IBM Systems Journal, 13 (2), 115—139, 1974.
7. Тимоти Бадд Объектно-ориентированное программирование в действии = An Introduction to Object-Oriented Programming. — СПб.: «Питер», 1997. — 464 с. — (В действии). — ISBN 5-88782-270-8.
8. Стив Макконнелл Совершенный код = Code Complete. — 2-е издание. — М.: Русская редакция, 2010. — С. 163-166. — 896 с. — (Мастер-класс). — ISBN 978-5-7502-0064-1

Математическая статистика и программная инженерия

1. Петрушин В.Н, Ульянов М.В. Информационная чувствительность компьютерных алгоритмов. — М.: ФИЗМАТЛИТ, 2010. — 224 с. [стр. 138–163].
2. Ульянов М.В., Наумова О.А., Яковлев И.А. Прогнозирование временных оценок для табличного алгоритма решения задачи оптимальной упаковки на основе функции трудоёмкости // Бизнес-Информатика 2008. №3(5). С. 37-46.
3. Головешкин В.А., Петрушин В.Н., Ульянов М.В. Количественные оценки информационной чувствительности алгоритмов // Информационные технологии и вычислительные системы. 2011. №4. С. 45-57.
4. Ульянов М.В., Петрушин В.Н., Кривенцов А.С. Доверительная трудоемкость – новая оценка качества алгоритмов // Информационные технологии и вычислительные системы. 2009. №2. С. 23 – 37.