МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
(Савченко А.С.)
«_____» _____ 2021р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"МАГІСТРА"

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: "Технологія верифікації та валідації програмних систем критичного призначення"

Виконавець: *Єрмолаєв Андрій Павлович*.

Керівник: *к.т.н., доцент, Райчев Ігор Едуардович*.

Нормоконтролер: *Райчев І.Е.*

Київ – 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *Кібербезпеки, комп'ютерної та програмної інженерії*

Кафедра *Комп'ютерних інформаційних* технологій

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Савченко А.С.
«____»_____2021р.

ЗАВДАННЯ
на виконання дипломної роботи студента

Єрмолаєва Андрія Павловича
(прізвище, ім'я, по батькові)

1. Тема роботи: «Технологія верифікації та валідації програмних систем критичного призначення» затверджена наказом ректора від «12» жовтня 2021р. за № 2228/ст.

2. Термін виконання роботи: з 12.10.2021р. по 31.12.2021р.

3. Вихідні дані до роботи: опис програмних систем критичного призначення; верифікація; технології тестування; стратегії та методи тестування ПЗ.

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): верифікація критичних програмниих систем; технологія тестування критичних програмних систем; валідація програмних систем критичного призначення; тестування модулів; інтеграційне тестування; звіт про життєвий цикл тестування.

5. Список графічних матеріалів: стратегія тестування програмних модулів автоматизованої системи контролю польотів; діаграма класів об'єктів тестування; стратегія тестування підсистем програм контролю польотів; зведений звіт тестування автоматизованої системи контролю польотів.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Підпис керівника
1.	Проаналізувати літературу та джерела за темою дипломної роботи. Розроблення та затвердження плану дипломної роботи	12.10.2021 - 14.10.2021	
2.	Провести консультації з науковим керівником щодо створення розділів 1, 2, 3	15.10.2021 – 16.10.2021	
3.	Розробити Розділ 1 « Принципи верифікації та валідації програмних систем критичного призначення»	17.10.2021 – 27.10.2021	
4.	Розробити Розділ 2 «Тестування та побудова стратегії тестування критичних програмних систем»	28.10.2021 – 10.11.2021	
5.	Розробити Розділ 3 «Використання технології тестування критичних програмних систем»	11.11.2021 – 07.12.2021	
6.	Висновки та оформлення пояснювальної записки дипломної роботи, підготовка доповіді та презентації дипломної роботи. Друк пояснювальної записки ДР	08.12.2021 – 15.12.2021	
7.	Підписати необхідні документи у встановленому порядку, підготовка до захисту дипломної роботи та попередній захист дипломної роботи на випусковій кафедрі	16.12.2021 – 20.12.2021	

7. Дата видачі завдання: «12» жовтня 2014 р.

Керівник дипломної роботи _____ Райчев І.Е.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Єрмолаєв А.П.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи "Технологія верифікації та валідації програмних систем критичного призначення" складається зі вступу, трьох розділів, висновку, списку використаних джерел і містить 106 сторінок, 18 рисунків, 22 таблиці. Список використаних джерел складається з 20 найменувань.

Ключові слова: КРИТИЧНІ ПРОГРАМНІ СИСТЕМИ, ТЕСТУВАННЯ, ВЕРИФІКАЦІЯ ТА ВАЛІДАЦІЯ, СТРАТЕГІЇ ТЕСТУВАННЯ, ТЕХНОЛОГІЇ ТЕСТУВАННЯ, МЕТОДИ ТЕСТУВАННЯ.

Об'єкт дослідження: критичні програмні системи та методи їх тестування.

Мета проекту: побудова стратегії тестування програмних систем критичного призначення, створення відповідної технології.

Метод проектування: аналіз відомих стратегій та вибір серед них такої, яка буде найповніше задовольняти вимоги до тестування програмних систем критичного призначення, що буде досліджено на прикладі тестування автоматизованих систем контролю польотів.

Матеріали дипломної роботи рекомендуються до використання при тестуванні різних класів програмних систем.

Галузь застосування дипломної роботи – тестування та верифікація програмних систем. В дипломній роботі розроблено технологію верифікації та тестування програмних систем критичного призначення.

– –

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

АСКП – автоматизована система контролю польотів

АП – аналоговий параметр

РК – разова команда

БР – бортовий реєстратор

ЛА – літальний апарат

КП – контроль польотів

ПІ – польотна інформація

ПЗ КП – програмне забезпечення контролю польотів

ІС – інформаційна система

ОС – операційна система

ПЗ – програмне забезпечення

ПС – програмна система

ЖЦ – життєвий цикл

ТНД – тестові набори даних

СУБД – система управління базами даних

V&V – верифікація та валідація

SVVP – Software Validation & Verification Plan – план верифікації та валідації програмного забезпечення

CM – Configuration Management – управління конфігурацією

SQA – Software quality assurance – забезпечення якості програмного продукту

CASE – Computer-Aided Software Engineering – набір інструментів та методів програмної інженерії

ВСТУП

Завжди актуальною є проблема підвищення безпеки функціонування об'єктів, контрольованих за допомогою ІС критичного призначення. Одним із шляхів досягнення цієї мети є забезпечення необхідного рівня якості ПС, які являють собою ядро кожної ІС. Це особливо важливо підчас експлуатації автоматизованих систем контролю, оскільки вони широко використовуються в авіації, енергетиці та інших галузях народного господарства і є критичними, бо пов'язані з безпекою життєдіяльності суспільства в цілому.

Наслідки низької якості розглянутих систем можуть бути досить серйозні і не раз призводили до аварій і катастроф (наприклад, аварії під час виконання польотів літальних апаратів, аварії на АЕС тощо). У зв'язку з цим, до характеристик ПЗ таких систем висуваються особливо жорсткі вимоги, які задаються у відповідних галузевих стандартах та інших нормативних документах. Для забезпечення необхідного рівня експлуатаційних характеристик критичних ПС проводиться атестація та сертифікація.

Програмне системи критичного призначення являють собою складні програмні комплекси, що придатні для використання у спеціалізованих обчислювальних системах і мають низку загальних істотних характеристик:

1) наявність загальних цілей і набору обов'язкових задач, що підлягають рішенню;
2) велика кількість елементів, що складають ПС (програми, модулі);
3) можливість виділення підсистем, які формуються з найбільш близьких по функціональних цілях груп елементів;
4) ієрархічна структура зв'язків між підсистемами;
5) наявність інтерактивних режимів роботи.

В якості прикладу, у дипломі будемо розглядати АСКП. ПС складаються із сотень модулів, що взаємодіють у процесі вирішення цільової задачі, якою є обробка параметричної інформації з метою прийняття діагностичних і управляючих рішень про стан об'єкта контролю та якість його функціонування, і має весь набір властивостей складних програмних систем.

Такі системи відносяться до критичних систем цільового призначення, а тому перед допуском до експлуатації необхідно виконувати незалежний контроль рівня їх якості, тобто оцінювати множину властивостей ПС шляхом сертифікаційних випробувань. Згідно стандартів термін сертифікація відповідності саме й означає дії третьої сторони (органа сертифікації та випробувальної лабораторії), спрямовані на підтвердження того, що ПС відповідає встановленим вимогам стандартів та інших нормативних документів.

Розділ 1. Принципи верифікації та валідації програмних систем критичного призначення

1.1. Опис критичних систем

Відомо, що комп'ютерні системи мають недоліки, тобто без явних причин інколи виходять з ладу, і не завжди ясно, що потрібно для відновлення їх працездатності. Програми, що виконуються на таких комп'ютерах, працюють невірно, деколи спотворюючи дані.

Функціональну надійність комп'ютерних систем можна визначити мірою довіри до них, тобто упевненістю, що система працюватиме так, як передбачається, і що збоїв не буде. Цю властивість не можна оцінити кількісно. Для цього використовуються такі відносні терміни, як "ненадійні", "дуже надійні" або "наднадійні", такі, що відображають різну міру довіри до системи.

Існує чотири основні складові функціональної надійності програмних систем (рис. 1.1), неформальні визначення яких приведені нижче.

1. Працездатність – властивість системи виконувати свої функції у будь-який час експлуатації.

2. Безвідмовність – властивість системи коректно (так, як чекає користувач) працювати весь заданий період експлуатації.

3. Безпека – властивість системи, що гарантує, що вона безпечна для людей і довкілля.

4. Захищеність – властивість системи протистояти випадковим або навмисним вторгненням в неї.

Працездатність і безвідмовність систем – показники, що носять імовірнісний характер і можуть бути виражені кількісно. Безпека і захищеність рідко виражаються у вигляді числових показників, але їх можна порівнювати за відносною шкалою рівнів. Наприклад, безпека рівня 1 менше безпеки рівня 2, яка, у свою чергу, менше безпеки рівня 3, і так далі.

Додаткові заходи, що підвищують функціональну надійність системи, можуть різко збільшувати вартість її розробки. Експоненціальний характер залежності "вартість-надійність" не дозволяє говорити про можливість створення ПС із стовідсотковою надійністю, оскільки вартість їх створення була б дуже великою.

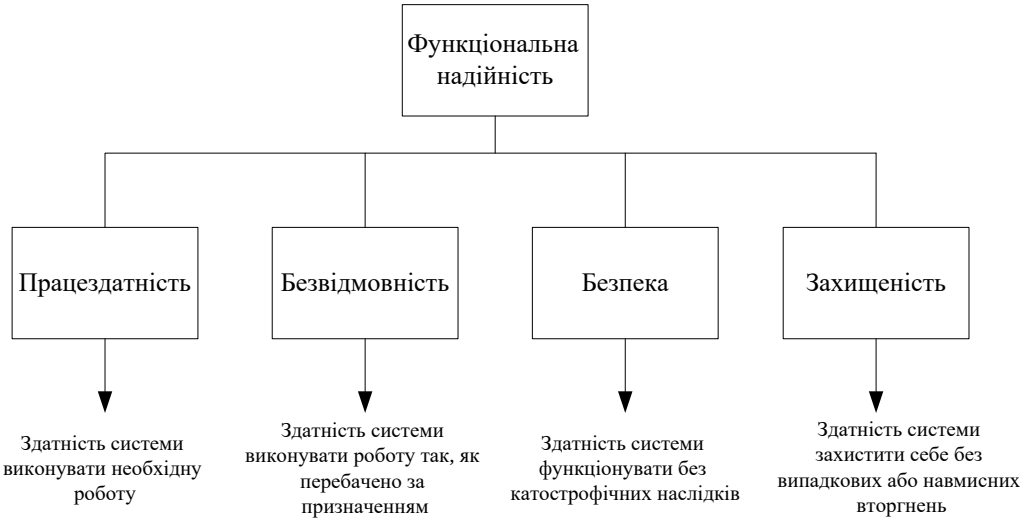

Рис. 1.1. Складові надійності системи

Високі рівні функціональної надійності можуть бути досягнуті лише за рахунок зменшення ефективності роботи системи. Наприклад, надійне програмне забезпечення передбачає додаткові, часто надлишкові, коди для перевірки нештатних станів системи. Це ускладнює систему і збільшує об'єм пам'яті, необхідний для її ефективної роботи. Але у ряді випадків надійність важливіша, ніж ефективність системи.

- Ненадійні системи часто залишаються незатребуваними. Якщо до системи немає довіри користувача, вона не буде затребуваною. Більш того, користувачі можуть відмовитися від інших програмних продуктів тій же компанії-розробника, оскільки будуть також вважати їх ненадійними.

- Вартість відмови системи може бути величезна. Для деяких застосувань, таких, як системи управління реакторами або системи навігації, вартість наслідків відмови може перевищувати вартість самої системи.

- Важко модернізувати ненадійну систему для підвищення її надійності. Зазвичай є можливість поліпшити неефективну систему, оскільки в цьому випадку основні зусилля будуть витрачені на модернізацію окремих програмних модулів. Систему, до якої немає довіри, важко поліпшити, оскільки ненадійність "розподілена" по всій системі.

- Існують можливості компенсувати недостатню ефективність системи. Якщо програмна система працює неефективно, то це постійний чинник, до якого користувач може пристосуватися, побудувавши свою роботу з його обліком. Ненадійність системи, як правило, виявляється раптово. Ненадійне програмне забезпечення може порушити роботу всієї системи, в яку воно інтегровано, і зруйнувати дані користувача без попередження, що може мати серйозні наслідки.

- Ненадійні системи можуть бути причиною втрати інформації. Збір і зберігання даних — дорога процедура, часто данні коштують більше, ніж комп'ютерна система, на якій вони обробляються. Дублювання даних для попередження їх втрати унаслідок ненадійності системи зажадає значних зусиль і фінансових коштів.

Надійність системи залежить від технології розробки ПЗ. Багаторазові тестування з метою виключення помилок сприяють розробці надійних систем. Проте немає простого зв'язку між якістю процесу створення і якістю готової системи.

1.1.1. Типи програмних систем критичного призначення

Зазвичай відмова систем, керованих з допомогою ПЗ викликає незручності, але вони не призводять до тривалих наслідків. Проте є системи, відмови яких можуть призводити до значних економічних втрат, фізичних пошкоджень, або створювати загрозу здоров'ю та навіть людському життю. Такі системи зазвичай називають критичними ПС. Функціональна надійність – необхідна вимога до критичних систем, і всі її складові (працездатність, безвідмовність, безпека і захищеність) дуже важливі. Не менш важливий для критичних систем і високий рівень надійності.

Існує три основні типи систем критичного призначення:

1. Системи, критичні по забезпеченню безпеки. Системи, відмова яких приводить до руйнувань, створює загрозу життю людини або завдає шкоди навколишньому середовищу. Як приклад можна привести систему управління виробництвом на хімічному заводі.

2. Системи, критичні для цільового призначення. Системи, відмова яких може призвести до помилок в діях, направлених на забезпечення певної мети. Прикладом може слугувати навігаційна система космічного корабля.

3. Системи, критичні для бізнесу. Відмова таких систем може завдати шкоди справі, в якій вони використовується. Прикладом є система, обслуговуюча рахунки клієнтів в банку.

Ціна помилки критичної системи часто дуже велика. Вона включає прямі витрати, пов'язані з внесенням змін до системи або її заміною, непрямі витрати, наприклад судові, і витрати, пов'язані з втратами в бізнесі. З високої можливої ціни відмови системи витікає, що якість методів розробки і сам процес створення ПЗ зазвичай важливіші, ніж вартість застосування цих методів.

Тому при створенні критичних систем зазвичай використовуються випробувані методи розробки, а не нові, такі, що ще не мали великого практичного вживання. Лише порівняно недавно такі відносно нові методи, як, наприклад, об'єктно-орієнтовані, стали використовуватися для розробки критичних систем, в той же час до цих пір при розробці багатьох критичних систем все ще застосовуються функціонально-орієнтовані методи.

З іншого боку, методи розробки ПЗ, які зазвичай нерентабельні, можуть використовуватися для розробки критичних систем, наприклад метод формальних специфікацій і формалізованої перевірки програм на відповідність таким специфікаціям. Однією з причин використання цих методів є зменшення кількості потрібного тестування. Для

критичних систем вартість перевірки і атестації зазвичай дуже висока і може складати більше 50% загальної вартості системи.

Необхідно відзначити, що функціональна надійність – це загальносистемне поняття. Розглядаючи надійність критичних систем, можна виділити три типи системних "компонентів", схильних до відмови:

- Апаратні засоби системи, що відмовляють або із-за помилок конструювання, або із-за помилок виготовлення, або із-за повного зносу;
- Програмне забезпечення системи, яке може відмовляти із-за помилок в технічних вимогах до системи, або в архітектурі системи, або в програмному коді;
- Людський чинник, який своїми діями порушує правильну роботу системи.

Таким чином, якщо мета полягає в тому, аби підвищити надійність системи, необхідно розглядати всі ці аспекти у взаємозв'язку.

1.1.2. Працездатність і безвідмовність

Дві близько зв'язані складові функціональної надійності – працездатність і безвідмовність. Під працездатністю системи мається на увазі здатність надавати користувачеві всі необхідні системні сервіси у міру виникнення потреби в них. Безвідмовність - це здатність системи надавати саме ті сервіси, які закладені в систему її специфікацією. Із цього слідує, що безвідмовність загальний показник, що включає властивість працездатності, оскільки якщо система не працездатна, то про безвідмовність взагалі не може йти мови.

Проте необхідно розрізняти ці властивості, оскільки вимоги до працездатності і безвідмовності в різних системах можуть бути різними. Наприклад, деякі системи можуть мати порівняно часті збої, але вони також можуть швидко відновлюватися після збоїв. В таких систем порівняно низькі вимоги до безвідмовності. В той же час вони можуть мати високі вимоги до працездатності у зв'язку з необхідністю безперервного обслуговування користувачів.

Інша відмінність між цими показниками полягає в тому, що працездатність системи залежить також від часу, який потрібний для усунення недоліків. Безвідмовність і працездатність системи можуть бути визначена точніше наступним чином.

- Безвідмовність - це здатність системи безвідмовно працювати певний час з вказаною метою в певному оточенні.
- Працездатність - це здатність системи правильно функціонувати і надавати вчасно необхідні сервіси.

Одна з практичних проблем, що виникає при розробці систем, полягає в тому, що наші інтуїтивні поняття безвідмовності і працездатності часто виявляються ширшими приведених вище формулювань. При розгляді функціональної надійності системи мають бути взяті до

уваги оточення, в якому буде діяти система, і мета, для якої вона використовується. Отже, оцінка безвідмовності в одному оточенні не обов'язково переноситься в інше оточення, де система використовується по-іншому.

Також важливі способи використання системи і реакція людини на її роботу. У визначенні працездатності і безвідмовності системи не розглядається тяжкість і наслідки відмов системи. Людям особливо небайдужі відмови, які мають серйозні наслідки; від цього залежить сприйняття безвідмовності системи.

Вважається, що система поводиться надійно, якщо її робота відповідає заданому алгоритму. Проте можливі ситуації, коли робота системи не зовсім відповідає очікуванням користувачів. На жаль, системні вимоги часто бувають або неповні або некоректні. Розробники в таких випадках повинні самі вирішувати, як поводитиметься система, але, не будучи фахівцями в конкретній області застосування системи, вони не можуть врахувати всі чинники і запрограмувати таку поведінку системи, яка необхідна користувачеві.

Вважається, що найбільш важливими складовими функціональної надійності є безвідмовність і працездатність. Якщо ж система ненадійна, то важко гарантувати її безпеку або захищеність. Ненадійність системи призводить до великих матеріальних втрат, такі системи набувають репутації неякісних і надалі втрачають довіру споживачів.

Безвідмовність системи визначається відсутністю збоїв. Відмови системи можуть відбуватись із-за поганого або неправильного її обслуговування, можуть бути наслідком помилок в алгоритмі, а можуть бути викликані несправностями систем зв'язку. Проте у багатьох випадках причиною помилкової поведінки системи є дефекти в самій системі. При розгляді безвідмовності необхідно розуміти відмінність в термінах збій, помилка і відмова, які визначені в таблиці 1.1.

Таблиця 1.1. Термінологія безвідмовності

Термін	Опис
Відмова системи	Припинення функціонування системи
Системні помилки	Помилкова поведінка системи, яка не відповідає її специфікації
Збій системи	Неправильне поводження
Помилка оператора	Невірні дії користувача, що викликали збій в роботі системи

Збої не обов'язково призводять до відмов системи, оскільки вони можуть бути короткочасними і система може прийти до нормального функціонування раніше, ніж станеться відмова. Системні помилки також не обов'язково призводять до відмов системи, оскільки системи мають захист, що гарантує, що помилковий режим буде виявлений і виправлений.

Термінологія, приведена в таблиці 1.1, допомагає зрозуміти три доповнюючих один одного підходи, використовуваних для підвищення безвідмовності систем.

- Запобігання збоям. Підхід до розробки ПЗ, що мінімізує можливість появи помилок і що виявляє помилки перш, ніж вони приведуть до збою системи (виявлення різних аномалій програмного коду).
- Виявлення помилок і їх усунення. Використання всіляких методів перевірки системи в різних режимах дозволяє виявити помилки і усунути їх до введення системи в експлуатацію. Регулярне тестування системи і її налагодження - приклад даного підходу.
- Стійкість до збоїв. Використання спеціальних методів, що гарантують, що помилки в системі не призведуть до збоїв і що збої не призведуть до відмови системи (засобів самовідновлення системи з використанням дублювання модулів).

Програмний збій призводить до відмови системи, коли "збійний" програмний код виконується над певними вхідними даними, що призводять до збою системи. При цьому програма може коректно працювати на інших вхідних даних. Програма обробляє вхідні дані, отримуючи вихідні дані. Деякі з вхідних даних призводять до відмов системи, в результаті програма генерує помилкові вихідні дані. Безвідмовність програмного забезпечення характеризується ймовірністю, з якою при виконанні програми серед множини вхідних даних зустрінуться такі, що призведуть до помилкових результатів обчислень.

Існує складний взаємозв'язок між безвідмовністю системи і кількістю прихованих програмних дефектів. Не всі програмні помилки в рівній мірі викликають відмову системи. Зазвичай в множині вхідних даних, що призводять до помилкових вихідних даних, є ряд даних, ймовірність вибору яких більше, ніж в інших даних. Якщо ці вхідні дані не вимагають для своєї обробки тієї частини ПЗ, яка містить помилки, то системних збоїв не буде. Таким чином, безвідмовність системи залежить переважно від кількості вхідних даних, що призводять до помилкових результатів під час нормальної експлуатації системи. Збої системи, які виявляються лише у виняткових ситуаціях, мало впливають на її надійність.

Надійність системи пов'язана з ймовірністю помилки, що виявляється під час експлуатації системи. Усунення програмних помилок в рідко використовуваних системних модулях мало вплине на підвищення безвідмовності системи. Наприклад, усунення 60% програмних помилок лише на 3% підвищить безвідмовність системи. Це підтверджується і дослідженнями помилок в програмних продуктах IBM. Багато помилок в програмних продуктах реально викликають збої системи після сотень або тисяч місяців експлуатації.

Отже, програма може містити помилки та все ж викликати довіру користувачів. Збої програми ніколи не виникатимуть, якщо не вибирати вхідних даних, що ведуть до збоїв. Крім того, досвідчені користувачі часто працюють, знаючи про помилки програмного

забезпечення, що викликають збої системи, і уміло уникають їх. Усунення помилок в таких випадках не дасть практично жодного підвищення надійності.

Кожен користувач системи по-своєму уникає "зустрічі" з системними помилками. Помилки, з якими зустрічається один користувач, можуть ніколи не зустрітися іншому.

1.1.3. Безпека

Безпека системи – це властивість, що відображає здатність системи функціонувати, не загрожуючи людям або довкіллю. Там, де безпека є необхідним атрибутом системи, говорять про систему, критичну по забезпеченню безпеки. Прикладами можуть служити контролюючі та управляючі системи, в авіації, системи управління процесами на хімічних і фармацевтичних заводах і системи управління автомобілями.

Управління систем, критичних по забезпеченню безпеки, набагато простіше організувати за допомогою апаратних засобів, ніж за допомогою ПЗ. Проте зараз будуються системи такої складності, що управління не може здійснюватися лише апаратними засобами. Із-за необхідності управляти великим числом сенсорів і виконавчих механізмів із складними законами управління потрібне управляюче програмне забезпечення. У якості прикладу можна привести системи управління військовими літаками. Вони вимагають постійного програмного керування літаком, що гарантує безпеку польоту.

ПЗ систем, що розглядаються в цьому розділі, підрозділяється на два класи:

- Первинне програмне забезпечення, критичне по критерію безпеки. Це ПЗ включається в систему у вигляді окремого блоку управління. Неправильна робота такого ПЗ може бути причиною відмови обладнання, унаслідок якого може виникнути загроза життю людини або нанесення шкоди довкіллю.

- Вторинне програмне забезпечення, критичне по критерію безпеки. Це ПЗ непрямим чином може призвести до непередбачених наслідків. Прикладами можуть служити автоматизовані системи в техніці, неправильна робота яких може привести до помилок в роботі об'єкту і поставити під загрозу життя людей. Інший приклад такого ПЗ – медична база даних, що містить опис ліків, призначених пацієнтам. Помилки в цій системі можуть привести до неправильного дозування препаратів.

Безвідмовність і безпека системи взаємозв'язані, але, вочевидь, цеє різні складові функціональної надійності. Звичайно, система, критична по забезпеченню безпеки, повинна відповідати своєму призначенню і функціонувати без відмов. Для забезпечення безперервного функціонування навіть в разі помилок вона повинна мати захист від збоїв. Проте відмовостійкість не гарантує безпеки системи. Програмне забезпечення може лише один раз спрацювати неправильно, і це призведе до нещасних випадків.

Не можна бути на сто відсотків упевненим, що системне програмне забезпечення безпечне і відмовостійке. Безвідмовність ПЗ не гарантує його безпеки з ряду причин.

- У системній специфікації може бути не визначена поведінка системи в деяких критичних ситуаціях. Високий відсоток збоїв систем – результат швидше за невірні або неповні вимоги, ніж помилки програмування.

- Збої в роботі апаратних засобів можуть привести до непередбачуваної поведінки системи, внаслідок чого програмне забезпечення стикається з непередбачуваною ситуацією. Коли системні компоненти близькі до стану відмови, вони можуть поводитися нестійко і генерувати сигнали, які можуть бути оброблені програмним забезпеченням непередбаченим чином.

- Оператори, що працюють з системою, можуть внести помилки, які в особливих ситуаціях здатні привести до збою системи.

При розробці систем, критичних по забезпеченню безпеки, використовується спеціальна термінологія, приведена в таблиці 1.2.

Таблиця 1.2. Термінологія безпеки

Термін	Опис
Аварія (або нещасний випадок)	Незапланована подія або послідовність подій, що призводять до людської смерті чи поранення; нанесення збитків майну або навколишньому середовищу. Приклад нещасного випадку – нанесення шкоди оператору машиною, яка управляється комп'ютерною системою.
Небезпека	Ситуації, при яких можливі нещасні випадки та аварії. Приклад небезпеки – відмова сенсора, котрий визначає наявність перешкоди попереду автомобіля.
Пошкодження	Оцінюється як збиток від небезпечних випадків. Пошкодження можуть бути як незначними, так і катастрофічними, що призводять до загибелі людей.
Серйозність небезпеки	Оцінюється по самим великим пошкодженням в результаті самих небезпечних випадків. Серйозність небезпеки може ранжуватись від катастрофічної, що призводить до загибелі людей, до незначної.
Ймовірність небезпеки	Ймовірність появи подій, що створюють небезпечні ситуації. Значення ймовірності визначається звичайним методом. Небезпечні події ранжуються від ймовірного до неможливого.
Ризик	Вимірюється як ймовірність того, що система буде причиною нещасного випадку. Для оцінки ризику визначається ймовірність небезпеки, серйозність небезпеки та ймовірність того, що небезпечна ситуація призведе до аварії.

Вважається, що система безпечна, якщо її експлуатація виключає аварії (нещасні випадки) або їх наслідки незначні. Цього можна досягти трьома доповнюючими один одного способами:

- Запобігання небезпеці. Система розробляється так, щоб запобігти небезпечним ситуаціям. Наприклад, аби під час експлуатації машини запобігти попадання рук оператора під лезо, в системі розкрою передбачається обов'язкове одночасне натиснення двох окремих кнопок управління.

- Виявлення і усунення небезпеки. Система розробляється так, щоб можливі небезпечні ситуації були виявлені і усунені до того, як вони приведуть до аварії.

- Обмеження наслідків. Система може включати способи захисту, мінімізуючи пошкодження, що виникають в результаті аварії, що сталася. Наприклад, в систему управління двигунами літака зазвичай включається автоматична система пожежогасіння. В разі спалаху така система дозволяє запобігти пожежі і не ставить під загрозу життя пасажирів і екіпажа.

Аварії і нещасні випадки зазвичай є результатом декількох подій, які відбуваються одночасно з непередбаченими наслідками. Аналізуючи серйозні аварії, видно, що майже всі вони сталися із-за комбінації системних збоїв, а не унаслідок окремих збоїв. Непередбачена комбінація збоїв призводила до відмови системи. Неможливо попередити всі комбінації збоїв системи і ці аварії – неминучий наслідок використання складних систем. Програмне забезпечення має тенденцію розростатися і ускладнюватися, а складність програмно-керованих систем збільшує вірогідність аварій і нещасних випадків.

Це, звичайно, не означає, що програмне управління обов'язково збільшує ризик, пов'язаний з системою. Програмне управління і поточний контроль можуть підвищити безпеку системи. Крім того, програмно-керовані системи можуть контролювати ширший діапазон умов у порівнянні, наприклад, з електро-механічними системами. Вони також досить легко набудовуються. Вони передбачають використання комп'ютерних засобів, яким властива висока надійність і які відносно компактні. Програмно-керовані складні системи можуть блокувати небезпеку. Вони можуть підтримувати управління в шкідливих умовах, зменшуючи кількість необхідного обслуговуючого персоналу.

1.1.4. Захищеність

Це здатність системи захищати себе від зовнішніх випадкових або навмисних дій. Прикладом зовнішніх дій на систему могли б бути комп'ютерні віруси, несанкціоноване використання системи, несанкціонована зміна системи або даних і так далі. Захищеність важлива для всіх критичних систем. Без прийнятного рівня захищеності працездатність, безвідмовність і безпека системи втрачають сенс, оскільки причиною пошкодження системи можуть бути зовнішні дії.

Це пов'язано з тим, що всі методи підтвердження працездатності, безвідмовності і захищеності покладаються на незмінність системи при експлуатації і на відповідність її параметрів спочатку встановленим. Якщо встановлена система була пошкоджена якимось чином (наприклад, якщо програмне забезпечення було змінене в результаті проникнення в систему вірусу), то параметри надійності і безпеки, які спочатку були закладені, не можуть більше підтримуватися. В цьому випадку програмне забезпечення може мати непередбачувані наслідки.

Є певні типи критичних систем, для яких захищеність, − найбільш важливий показник надійності системи. Військові системи, системи для електронної торгівлі і системи створення

та обміну конфіденційною інформацією, повинні розроблятися з дуже високим рівнем захищеності. Наприклад, якщо система резервування квитків авіакомпанії недоступна, це заподіює незручність у зв'язку з деякою затримкою продажу квитків, але якщо система не захищена і може приймати підроблені замовлення, то авіакомпанія може зазнати великих збитків. Існує три типи пошкоджень системи, які можуть бути викликані зовнішніми діями:

- Відмова в наданні системних сервісів. Система може бути переведена в такий стан, коли нормальний доступ до системних сервісів стає неможливим.
- Руйнування програм і даних. Компоненти програмного забезпечення системи можуть бути не санкціоновано змінені. Це може вплинути на поведінку системи, а отже, на надійність і безпеку. Якщо пошкодження серйозне, система може стати не придатною до експлуатації.
- Розкриття конфіденційної інформації. Інформація, що знаходиться під управлінням системи, може бути конфіденційною, зовнішнє проникнення в систему може зробити її публічно доступною. Залежно від типу даних, це може вплинути на безпеку системи і викликати подальші зміни в системі, які позначаться на її працездатності і безвідмовності.

Як і у випадку з іншими складовими надійності, є спеціальна термінологія, пов'язана із захищеністю систем. Деякі терміни приведені в таблиці 1.3.

Таблиця 1.3. Термінологія захищеності

Термін	**Опис**
Зовнішній вплив	Влив, у разі якого можлива втрата даних і/або пошкодження системи
Вразливість	Дефект системи, котрий може стати причиною втрати даних чи її пошкодження
Атака	Використання вразливості системи
Загрози	Обставини, які можуть призвести до втрати даних чи пошкодження системи
Контроль	Захисні заходи, що зменшують вразливість системи

У термінології захищеності є багато загального з термінологією безпеки. Так, зовнішня дія аналогічна аварії (нещасному випадку), а вразливість – небезпеці. Отже, існують аналогічні підходи, що збільшують захищеність системи:

- Запобігання вразливості. Система розробляється так, щоб її вразливість була якнайнижче. Наприклад, система не з'єднується із зовнішньою мережею, аби уникнути дії з неї.
- Виявлення і усунення атак. Система розробляється так, щоб виявити зроблену на неї атаку і усунути її, поки вона не привела до пошкоджень і втрат. Приклад виявлення атак і їх усунення – використання антивірусних програм, які аналізують інформацію, що надходить, на наявність вірусів і усувають їх в разі проникнення в систему.

- Обмеження наслідків. Система розробляється так, щоб звести до мінімуму наслідки зовнішньої дії. Наприклад, регулярна перевірка системи і можливість переустановити її в разі пошкодження.

Захищеність стає ще актуальнішої при підключенні системи до Internet. І хоча Internet-зв'язок забезпечує додаткові функціональні можливості системи (наприклад, клієнт може отримати віддалений доступ до свого банківського рахунку), така система може бути зруйнована зловмисниками. При підключенні до Internet вразливі місця системи стають доступними для великої кількості людей, котрі можуть впливати на систему.

Дуже важливим атрибутом систем, підключених до Internet, є їх життєздатність, тобто здатність системи продовжувати працювати, тоді як вона піддається зовнішнім діям і частина її пошкоджена. Життєздатність, звичайно, пов'язана і із захищеністю, і з працездатністю. Для забезпечення життєздатності слід визначити основні ключові компоненти системи, які необхідні для її функціонування. Для підвищення життєздатності використовується три стратегії: протидія зовнішнім діям, розпізнавання їх і відновлення системи після атаки.

1.2. Процес перевірки в життєвому циклі ПС. Призначення процесів перевірки

Разом з процесом забезпечення гарантії якості, в архітектурі процесів ЖЦ визначено ще чотири процеси підтримки, цілком направлених на підвищення якості ПС. Це процеси Верифікації, Валідації, Спільного перегляду і Аудиту, які призначені для перевірки відповідності робочих продуктів і процесів розробки своєму призначенню і підтвердження їх спроможності забезпечити належну якість кінцевого програмного продукту.

Виходячи з положень стандарту ISO/IEC 12207 «призначення процесу верифікації полягає в підтвердженні того, що кожен робочий продукт ПС (software work product) і послуга процесу або проекту належним чином відображають встановлені вимоги. В результаті успішного здійснення процесу:

- буде розроблена і впроваджена стратегія верифікації;
- будуть встановлені критерії верифікації всіх необхідних робочих продуктів ПС;
- виконуватимуться належні дії з верифікації;
- виконуватиметься пошук дефектів в робочих продуктах і їх усунення;
- забезпечуватиметься доступ замовника і інших зацікавлених сторін до результатів діяльності верифікації.

Призначення процесу валідації полягає в підтвердженні того, що вимоги, що стосуються конкретного вживання робочого продукту ПС за призначенням, задовольняються. В результаті успішного здійснення процесу:

- буде розроблена і впроваджена стратегія валідації;
- будуть ідентифіковані критерії валідації для всіх потрібних робочих продуктів;

- проводитиметься належна валідаційна діяльність;
- вирішуватимуться всі виявлені проблеми;
- надаватимуться свідоцтва того, що розроблені робочі продукти ПС відповідають своєму призначенню;
- забезпечуватиметься доступність результатів валідаційної діяльності для замовника і інших зацікавлених сторін».

Аналізуючи визначення стандарту досить складно встановити межу між процесами верифікації і валідації, оскільки обидва процеси застосовуються до «робочих продуктів ПС» (software work product). У стандарті ISO/IEC 12207 (1995), що діє, ця межа проведена чітко: поняття верифікації зв'язане з програмними продуктами (software products) діяльності по розробці ПС а валідації – з «кінцевим програмним продуктом» (final software product).

На практиці процеси верифікації і валідації зазвичай виконуються спільно. Вони визначені в стандарті окремо швидше з міркувань дотримання «принципу модульності» архітектури процесів.

Призначення процесу спільного перегляду, відповідно до ISO/IEC 12207, полягає в тому, аби «підтримувати належний рівень розуміння замовником того, як проект просувається до цілей договору, і того, які саме додаткові заходи мають бути проведені для забезпечення гарантії розробки продукту, який його задовольнить. Спільні перегляди проводяться як на рівні управління проектом, так і на технічному рівні, і протягом всього життєвого циклу проекту. В результаті успішного здійснення процесу:

- періодичні перегляди виконуватимуться в заздалегідь визначені терміни;
- стан і продукти діяльності процесу оцінюватимуться в ході виконання спільного перегляду за участю замовників, постачальників і інших відповідальних осіб і користувачів;
- результати перегляду доводитимуться до всіх сторін, яких вони стосуються;
- заходи, рішення про виконання яких прийняте при переглядах, відстежуватимуться до їх завершення;
- проблеми ідентифікуватимуться і вирішуватимуться».

В ході підготовки спільного перегляду мають бути визначені сфера перегляду, теми, учасники, графік роботи і вимоги до ресурсів, і встановлені критерії перегляду. Критерії перегляду стосуються способу виявлення проблеми і її рішення, а також узгодження результату перегляду його учасниками.

Мета періодичних спільних переглядів на рівні управління проектом – оцінити стан проекту по відношенню до затверджених планів, графіків, стандартів і керівництва, встановити, чи немає необхідності в зміні планів, чи правильно розподілені ресурси, чи виконується управління ризиками.

Мета періодичних спільних переглядів на технічному рівні – оцінити стан робочих продуктів ПС по відношенню до вимог замовника і критеріїв приймання, зафіксованих в договорі. В ході технічних переглядів потрібно встановити, чи ведеться розробка робочих продуктів за планом, чи відповідає міра їх завершеності запланованим, чи відповідають вони стандартам і специфікаціям, чи всі зміни в них вносяться належним чином і стосуються лише тих областей, які визначені процесом управління конфігурацією.

Мета внутрішніх періодичних спільних переглядів процесу - оцінити його спроможність і придатність для проекту. Проблеми, виявлені при перегляді, упорядковуються по мірі важливості для проекту і фіксуються в звіті разом з пропонованими заходами по їх рішенню. Звіт прямує на узгодження всім учасникам перегляду і потім передається процесу вирішення проблем. Виконання заходів контролюється.

Призначення процесу аудиту, згідно ISO/IEC 12207, полягає в незалежному встановленні відповідності вибраних продуктів і процесів вимогам, планам і договору. В результаті успішного здійснення процесу:

- буде розроблена і впроваджена стратегія аудиту;
- аудити проводитимуться;
- узгодженість вибраних робочих продуктів ПС і послуг або процесів з вимогами, планами і договором встановлюватиметься відповідно до стратегії аудиту;
- проблеми, виявлені під час аудиту, будуть ідентифіковані, доведені до тих, хто несе відповідальність за заходи щодо коректування, і вирішення».

Стандарт визначає робочі продукти і результати діяльності по процесам ЖЦ. що підлягають аудиту:

- відповідність коду програмного продукту проектній документації;
- адекватність вимог до приймальних переглядів і випробувань, зафіксованих в документації, прийнятті програмного продукту;
- відповідність тестових даних специфікації;
- успішність випробувань програмного продукту і його відповідність специфікації;
- коректність звітів про випробування;
- усунення розбіжностей між фактичними і очікуваними результатами;
- відповідність документації користувача встановленим стандартам;
- відповідність виконаних дій застосованим до них вимогам, планам і договору;
- відповідність витрат і термінів плановим.

Аудит проводиться в заздалегідь встановлені терміни. При підготовці плану аудиту встановлюється сфера вживання аудиту, теми, учасники, критерії початку і завершення аудиту, графік і потрібні ресурси.

Окрім аудитів розробки ПС можуть також проводитися аудити управління проектом і аудити виконання процесів, з метою визначення їх спроможності і придатності для проекту. За результатами аудиту складається аудиторський звіт. У ньому фіксуються проблеми, виявлені в процесі аудиту, і пропоновані заходи щодо їх рішення. Звіт поширюється серед учасників аудиту і прямує процесу вирішення проблем. Виконання заходів контролюється.

Кожен з розглянутих вище процесів повинен синхронізуватися з основними процесами ЖЦ, для перевірки робочих продуктів до яких він підключається (у контрольних точках проекту), і координуватися з процесом SQA відносно видів і способів перевірок.

Рівень і масштаб планування і виконання перевірок варіюється в залежності від особливостей ПС і умов її розробки – вибраної моделі ЖЦ, вихідних вимог до ПС, що використовуються для організаційних підходів і процедур, розміру, критичності і типу ПС, кількості персоналу проекту і сторін, що беруть участь в проекті. Ці чинники визначають, які з п'яти процесів підтримки якості будуть включені в модель процесів ЖЦ проекту у вигляді окремих процесів, і які завдання з інших процесів додатково в них вирішуватимуться. Так, в простому випадку, для проекту може бути вибраний, наприклад, лише процес SQA, в який включені завдання процесу спільного перегляду, а процеси верифікації і валідації, а також аудиту - не використовуватимуться. Проте для масштабних проектів розробки критичних ПС можуть знадобитися і незалежна верифікація і валідація (V&V), і періодичні спільні перегляди, і аудиторські перевірки третьою стороною.

Згадані вище чинники обумовлюють і вибір методів для вирішення завдань в даних процесах – залежно від специфіки проекту і вимог до продукту перевірки можуть бути індивідуальними або колективними, проводитися колегами за проектом або незалежними експертами і відрізнятися формою організації робіт. Одні і ті ж методи можуть використовуватися в різних процесах. Стандарт ISO/IEC 12207 вказує лише на те, **що** (які дії) повинні виконуватися в рамках процесів, але не на те, **як** (якими методами) їх виконувати. Часто виникає плутанина через наявність однойменних процесів і методів, а також неточних перекладів деяких термінів. Так процес верифікації інколи пов'язують виключно з методом формальної верифікації (доказом коректності), а процес спільного перегляду (joint review) плутають з методом колегіальної перевірки (peer review) чи технічного огляду (technical review). Методи перевірки, що розглядаються в цьому розділі, регламентуються наступними стандартами:

- IEEE Std. 1012 «IEEE Standard for Software Verification and Validation» (1998),
- IEEE Std. 1059 «IEEE Guide for Software Verification and Validation Plans» (1993),
- IEEE Std. 1012a «Supplement to IEEE Standard for Software Verification and Validation: Content Map to IEEE/EIA 12207.1-1997»,

• IEEE Std. 1028 «IEEE Standard for Software Reviews» (1997). Нова редакція стандарту 1988 р. «IEEE Standard for Software Reviews and Audits».

1.3. Мета і задачі верифікації та валідації

Роль процесів V&V полягає у виконанні перевірки і оцінюванні стану робочих продуктів ПС впродовж процесу розробки з метою своєчасного виявлення проблем проекту.

Тоді як SQA більшою мірою акцентує увагу на перевірці дотримання стандартів і дієвості процесів, V&V зосереджуються на технічних аспектах розробки, відбитих в документах або компонентах ПС (договорі, вихідних вимогах, проекті, коді, інтегрованою ПС). Мета V&V - перевірити і підтвердити, що вимоги, що пред'являються до ПС є повними, точними, погодженими, коректно встановленими і тестопридатним, а програмні робочі продукти коректно їх реалізують. Інфраструктура розробки ПС зображена на рисунку 1.2.

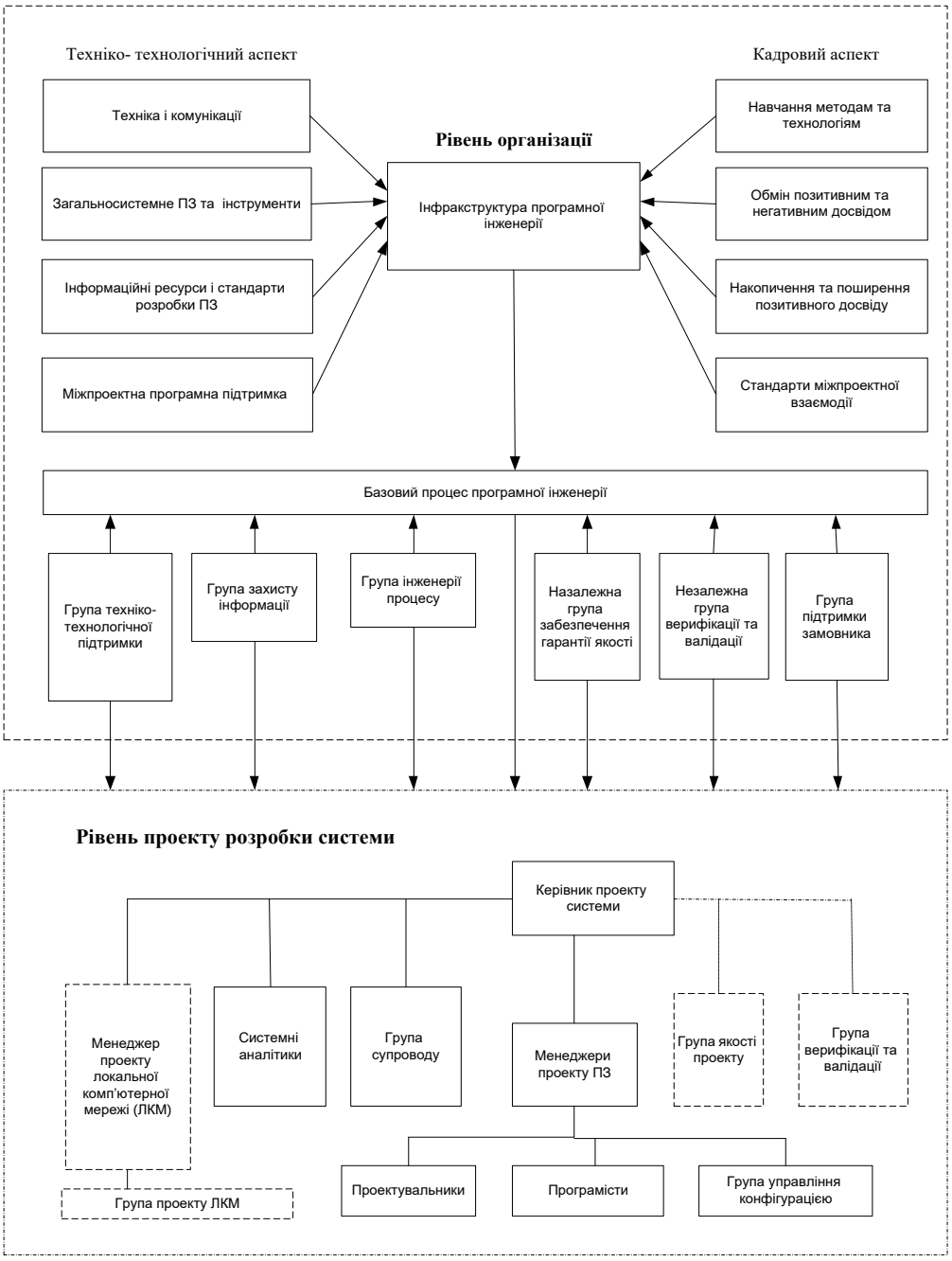

Рис. 1.2. Інфраструктура розробки програмних систем

Процеси V&V оцінюють елементи програмного забезпечення в контексті системи, частиною якої воно є, з врахуванням вимог з боку операційного оточення (середовища експлуатації), технічних засобів (hardware), інтерфейсів, обслуговуючого персоналу системи і безпосередніх користувачів програмних продуктів.

Склад об'єктів перевірки, критеріїв перевірки, завдань V&V, а також інтенсивність і ретельність виконання процесів V&V визначається рівнем цілісності системи, що розробляється (рівнями критичності її компонентів). Зв'язок і відмінність процесів верифікації і валідації. Суть відмінностей процесів верифікації і валідації зручніше сформулювати стосовно каскадної моделі ЖЦ і традиційних стадій розробки програмного забезпечення систем (визначення концепції, аналіз вимог, проектування, реалізація (побудова), інтеграція і тестування, введення в дію (інсталяція)).

Ролі спеціалістів в організаційній структурі розробки прописані в таблиці 1.4.

Таблиця 1.4 . Ролі спеціалістів в організаційній структурі розробки

Ролі на рівні організації	**Обов'язки**
Група захисту інформації	- Вивчення стану справ у сфері захисту інформації та накопичення досвіду - Забезпечення захисту інформації в організації - Перевірка захисту інформації в організації - Підтримка проектів у питаннях захисту інформації
Група інженерії процесу (технологічна служба)	- Виявлення, супровід та удосконалення базового процесу програмної інженерії. Забезпечення нормативно-методичної підтримки виконання процесів ЖЦ. Організація та поповнення сховищ (бібліотек) активів організації - Допомога менеджерам проекту в адаптуванні базового процесу до потреб проектів. Підбір чи виготовлення форм (шаблонів) документів для інженерії проектів - Підтримка процесу документування в проектах, при виконанні графічних робіт, оформленні документів у відповідності до стандартів оформлення. Нормоконтроль та друк документів - Міжпроектне координування в частині накопичення досвіду та організації навчання - Підтримка та управління конфігурацією в проектах (за домовленістю з керівниками проекту)
Незалежна група якості (SQA-група)	- Планування та виконання дій по контролю і гарантуванню дотримання дисципліни створення програмної продукції в проектах (організація перевірок робіт в контрольних точках проекту, визначених календарним планом) - Контроль документів і продуктів ПЗ в контрольних точках проектів на предмет дотримання діючих стандартів та інших нормативних документів, встановлених у вимогах замовника - Звітність безпосередньо перед керівником організації
Незалежна група верифікаціх та валідації (V&V-група)	- Виконання функцій верифікації (за домовленістю з групою SQA) - Планування та проведення незалежного кваліфікованого тестування інтегрованих компонентів ПЗ або програмних продуктів з метою визначення їх відповідності вимогам замовника - Координування планів робіт з менеджерами проектів відносно вимог до тестового середовища, термінів та порядку передачі ПЗ на

	тестування - Надання звітів тестування менеджерам проектів для прийняття мір по виправленню ПЗ - Незалежність від менеджерів проекту в частині визначення об'ємів та методів тестування - Звітність перед керівником організації за дотримання порядку тестування та стан розроблених програмних продуктів
Група підтримки замовника	- Зв'язок з замовником по питанням автоматизації ділових процесів - Підтримка процесів управління вимогами, навчання користувачів, підтримка
Ролі на рівні проекту	**Обов'язки**
Керівник проекту системи	- Повна фінансова відповідальність за виконання проектних домовленостей перед замовником - Управління розробкою складових продукції, що створюється – продуктів ПЗ, засобів захисту інформації - Відповідальність за дії учасників проекту
Системні аналітики	- Перевірка умов та …. автоматизації діяльності організації-споживача - Системний аналіз вимог споживача та формування концепції системи - Контроль обґрунтування проектних рішень, що приймаються
Група якості проекту	- Контроль якості робочих продуктів, вироблених процесами ЖЦ - Підзвітність лише керівникові проекту
Група V&V проекту	- Перевірка відповідності робочих продуктів, створених на певному етапі ЖЦ, вимог до них, встановлених на попередньому етапі - Може виконувати тестування окремих компонентів ПЗ, а також системне (інтеграційне) тестування ПЗ, виконаного у проекті - Підзвітна лише керівникові проекту
Менеджер проекту ПЗ	- Повна відповідальність за всі проектні рішення та дії, зв'язані з розробкою ПЗ в проекті - Підбір та контроль ресурсів проекту, а також графік робіт - У великих чи розподілених програмних проектах може бути декілька менеджерів
Проектувальник	- Прийняття та документування програмних рішень по архітектурі та функціям ПЗ. Узгодження рішень з менеджером проекту ПЗ - Дотримання стандартів якості
Програмісти	- Програмування або моделювання (макетування, швидке прототипування) компонентів ПЗ по проектним специфікаціям (постановка задач), підготовлених проектувальниками - Дотримання стандартів якості при програмуванні - Налагодження та автономне тестування розроблених компонентів
Група управління конфігурацією	- Виконання процесу конфігураційного управління версій ПЗ та робочих продуктів проекту ПЗ
Група супроводу	- Виконання процесу супроводу версій ПЗ та робочих продуктів проекту ПЗ під час експлуатації та протягом встановленого періоду супроводу - Навчання користувачів - Виконання процесу вирішення проблем - Можуть бути в команді групи підтримки замовника

Верифікація означає перевірку відповідності конкретних вихідних робочих продуктів кожної стадії вимогам до них, встановленим на попередній стадії. Оцінюється коректність, повнота і точність реалізації вимог, а також відповідність стандартам, що діють, і нормам. Верифікація виконується з метою своєчасного виявлення і усунення допущених помилок і забезпечення керівництва проекту об'єктивними відомостями, необхідними для управління

ризиком проекту. Результати верифікації служать базисом для оцінки завершеності поточної стадії і ухвалення рішення про перехід до наступної стадії проекту.

Валідація означає підтвердження того, що (ті самі) робочі продукти ПЗ задовольняють системним вимогам, делегованим ПЗ (тобто, тій частині вимог до системи, яка буде реалізована програмними продуктами), і відповідають потребам системи (наприклад, алгоритми коректно моделюють фізичні закони системи: або, моделі даних, функцій, станів і процесів адекватні діловим процесам банківської системи).

Якби вихідні вимоги до програмного забезпечення системи повністю охоплювалися на початку проекту і були незмінні (що і передбачається при виборі каскадної моделі розробки), процес валідації можна було б зв'язувати лише з вихідними вимогами, тестами і кінцевим програмним продуктом. В цьому випадку була б виправданою існуюча думка про те, що валідація ототожнюється з тестуванням при приймальних випробуваннях. Проте, на практиці, вимоги змінюються в ході розробки. Та і сучасні CASE-середовища дозволяють зберігати і «тестувати» (трасувати до вихідних вимог) проміжні продукти розробки.

Таким чином, верифікація, і валідація необхідні і можливі на всіх стадіях створення ПЗ. Питання полягає в правильному оцінюванні рівня цілісності ПЗ і належного об'єму V&V.

V&V підтримує основні процеси ЖЦ і, як вказується в стандарті IEEE Std. 1012, «найбільш ефективна, коли застосовується паралельно з процесами розробки ПЗ; інакше цілі V&V можуть не бути досягнуті. У цьому стандарті процеси V&V розглядаються спільно, оскільки їх дії і завдання перетинаються і взаємно доповнюють один одного». Має бути виявлене коло основних процесів в проекті, з якими асоціюватиметься V&V і для них вибрана мінімально необхідна множина задач перевірки, що визначається рівнем цілісності, встановленим для програмного забезпечення системи.

Рівні цілісності програмного забезпечення. Рівні цілісності ПЗ системи зв'язуються з діапазоном значень критичних характеристик ПЗ (надійності, безпеки функціонування, захисту інформації, продуктивності, складності і ін.), які дозволяють утримувати ризики в прийнятних межах. Критичне високо-цілісне ПЗ вимагає V&V, більшої за об'ємом і суворістю, ніж не критичне.

Рівні цілісності можуть призначатися для вимог до програмного забезпечення, функцій, груп функцій, програмних компонентів або підсистем на первинних стадіях проекту, узгоджуватися всіма зацікавленими сторонами і змінюватися по мірі еволюції розробки у зв'язку з вибором організацією-розробником або замовником певних стратегій і методологій розробки і управління ризиком. Контроль і зміна рівня цілісності – завдання аналізу критичності (V&V criticality analysis) в ході процесу розробки.

Стандарт використовує 4-рівневу схему цілісності ПЗ як метод для визначення мінімуму завдань V&V в рамках рівня (таблиця 1.5).

Таблиця 1.5. Рівні цілісності ПЗ

Критичність	Опис	Рівень
Висока	Вибрана функція слугує критичним фактором для ефективної роботи системи	4
Велика	Вибрана функція важлива для ефективної роботи системи	3
Помірна	Вибрана функція впливає на роботу системи, але можуть бути реалізовані стратегії обходу для компенсації втрати ефективності	2
Низька	Вибрана функція робить вагомий внесок в роботу системи. Якщо вона не буде виконуватись належним чином, користувач відчує незручності	1

Якщо рівень цілісності ПС або окремих компонентів програмного забезпечення системи нижче першого, вживання V&V для них необов'язково.

Асоціація порівневої схеми цілісності з мінімумом завдань V&V описується в Плані верифікації і валідації (SVVP), а обґрунтування призначення певних рівнів цілісності компонентам, що перевіряються, ПЗ – в Звіті про завдання V&V і в Завершальному звіті про V&V.

Потрібно відмітити, що попередня редакція стандарту IEEE Std. 1012 (1986) була «продукто-орієнтованою» і визначала структуру і вміст SVVP. Нині чинний стандарт підтримує процесо-орієнтований підхід до розробки ПС і визначає процеси V&V в термінах дій і завдань, а також встановлює вимоги до управління V&V, плану SVVP і звітним документам.

Завдання V&V на стадіях розробки програмної системи. У стандарті IEEE Std. 1012 приведений перелік і опис мінімального набору завдань V&V для всіх основних процесів і рівнів цілісності програмного забезпечення систем. На рисунку 1.5 показані лише завдання процесу розробки разом з вхідною інформацією для їх вирішення (входами V&V) і вихідною інформацією – результатом їх рішення (виходами V&V), а в таблиці 1.5 коротко описані ті ж завдання з вказівкою номерів рівнів цілісності ПЗ, на яких вони вирішуються.

Враховуючи те, що завдання по V&V, які повинні вирішуватися для ПС з низькою критичністю (перший рівень цілісності), представляють найбільш широкий інтерес, ми розглядаємо їх далі детальніше.

Завдання верифікації і валідації вимог до ПС з низькою критичністю. Вимоги до ПС відбиваються в документах описі концепції ПС, специфікації вимог до ПС, а також специфікації вимог до інтерфейсів ПС. В ході розгляду цих документів вирішуються два завдання верифікації і валідації: оцінка вимог до ПС і створення всрифікація плану тестування для V&V системи.

Оцінка вимог до ПС полягає в аналізі всіх видів вимог з метою перевірки і підтвердження їх коректності, узгодженості, повноти, точності, читабельності і тесто-придатності. Підтвердження коректності вимог до ПС має бути засноване на:

- перевірці, чи правильно розподілені функції ПС за компонентами загально-системного і спеціального (що розробляється) програмного забезпечення, і чи задовольняють вони вимоги замовника до ПС;

- перевірці, чи узгоджуються вимоги до ПС із стандартами, що діють в галузі, нормами, положеннями і правилами ведення бізнесу;

- аналізі логіки подій і потоків даних в діловому процесі і підтвердженні, що послідовності зміни станів системи відповідають принципам, що діють в проблемній області, і іншим нормам;

- перевірці, чи задовольняють описи потоків даних і управління в ПС вимогам до її функцій і працездатності;

- перевірці правильності використання (обробки) даних і їх форматів.

Задачі V&V для процесів розробки зображені на рисунку 1.3.

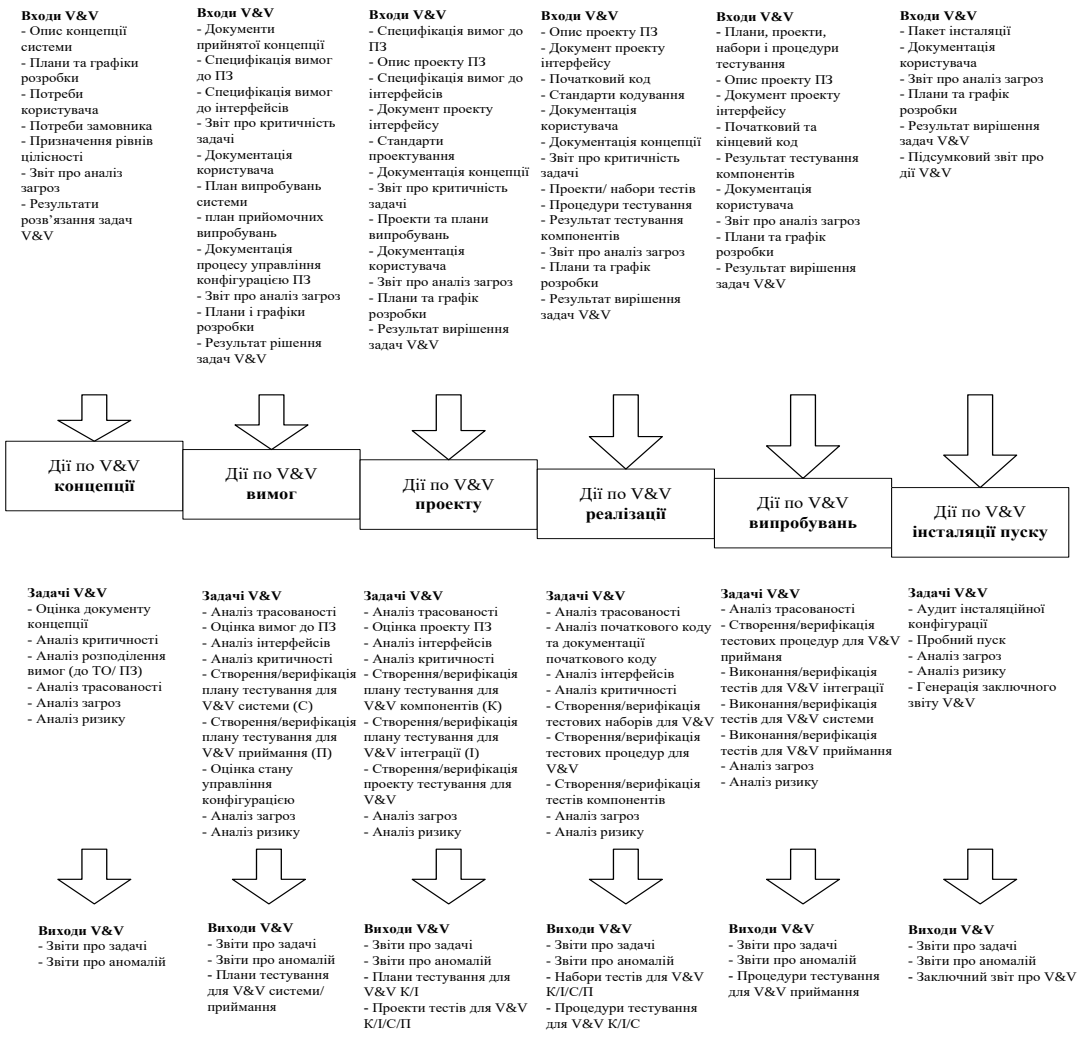

Рис. 1.3. Задачі V&V для процесів розробки

Характеристика задач V&V приведена у таблиці 1.6.

Таблиця 1.6. Характеристика завдань V&V

Завдання	**Опис завдання**	**Рівень**
Аналіз критичності	Завдання виконується у складі робіт по V&V концепції вимог, проекту і реалізації ПЗ системи. Перевіряється коректність призначення рівнів цілісності вимогам, функціям, підсистемам, модулям ПЗ. Якщо приватні рівні цілісності не призначені – призначається рівень цілісності, призначений системним вимогам. Якщо якому-небудь компоненту призначений самий високий рівень критичності, то усім, що «впливає» на нього компонентам привласнюється той же рівень. Рівні цілісності документуються в звіті про виконання V&V.	4,3,2
Аналіз розподілу системних вимог	Завдання виконується у складі робіт по V&V концепції. Перевіряється коректність, точність і повнота розподілу системних вимог по компонентах технічного і програмного забезпечення, а також призначеним для користувача інтерфейсам в контексті вимог користувачів. Перевіряється, чи забезпечить схема розподілу потрібну продуктивність роботи, чи задовольняють формати даних, частота і протоколи передачі інформації користувацьким вимогам до інтерфейсів, передбачені механізми забезпечення відмово-стійкості і відновлюваності застосувань, чи виконуються вимоги користувачів до супроводу системи.	4
Аналіз відстеження	Завдання виконується у складі робіт по V&V концепції вимог, проекту, реалізації і випробувань ПЗ. Спочатку в документі опису концепції ідентифікуються усі системні вимоги, які повністю чи частково будуть реалізовані програмно. Перевіряється чи простежуються системні вимоги до вимог замовника. Критерій відстеження - коректність узгодженість в деталях, повнота і точність відображення вимог замовника у вимогах до системи. На подальших стадіях розробки перевіряється, відповідно, чи простежуються вимоги до ПЗ по системним вимогам (при V&V вимог), елементи проекту ПЗ до вимог до ПЗ (при V&V проекту), компоненти початкового коду відповідним специфікаціям в проекті і навпаки (при V&V реалізації), а все тестові процедури - до планів тестування (при V&V випробуваннях).	4,3,2
Аналіз загроз	Завдання виконується у складі робіт по V&V концепції вимог, проекту, реалізації, випробувань і вводу в дію ПЗ системи. Спочатку по документу опису концепції визначається чи існують потенційні загрози системі або з боку системи. Оцінюється вірогідність появи кожної загрози і її серйозність, ідентифікуються стратегії усунення загроз. Складається звіт про аналіз загроз. При наступних перевірках уточнюється, які вимоги до ПЗ (проектні рішення, особливості реалізації) вносять вклад в кожну з раніше виявлених загроз.	4, 3
Аналіз інтерфейсів	Завдання виконується у складі робіт по V&V вимог проекту і реалізації ПЗ системи. Спочатку за документами опису концепції, опис вимог до ПЗ і описи інтерфейсів перевіряється і підтверджується, що вимоги до інтерфейсів ПЗ з користувачем, оператором і іншими системами : - коректні. Правильно визначені вимоги до зовнішніх і внутрішніх інтерфейсів. - погоджені. Несуперечливі описи інтерфейсів у документі специфікації вимог і документі специфікації інтерфейсів	4, 3, 2

	- повні. Описані формати даних і критерії робото-спроможності кожного інтерфейсу - точні. Кожен інтерфейс забезпечує передачу інформації з потрібною точністю. - тесто-придатні. Існують об'єктивні критерії приймання для підтвердження вимог. При V&V проекту за тими ж критеріями перевіряються усі проектні рішення відносно інтерфейсів з ПЗ, користувачем, оператором і іншими системами. При V&V реалізації за тими ж критеріями перевіряється і підтверджується правильність програмно реалізованих інтерфейсів.	
Оцінка документу опису концепції	Завдання виконується у складі робіт по V&V концепції ПЗ системи. Перевіряється, наскільки документація концепції задовольняє вимоги користувачів і відповідає потребам замовника. Підтверджується, що функції системи, цикл її використання, міра тієї, що реалізовується і тесто-придатність, функціональні вимоги, концептуальний проект архітектури системи, вимоги до експлуатації і супроводу, а також вимоги до міграції напрацювань із старої системи в нову задовольняють користувачів.	4, 3, 2
Оцінка вимог до ПЗ	Завдання виконується у складі робіт по V&V вимог до ПЗ. Описана окремо	4,3,2,1
Оцінка проекту ПЗ	Завдання виконується у складі робіт по V&V проекту ПЗ. Описана окремо	4,3,2,1
Оцінка початкового коду	Завдання виконується у складі робіт по V&V реалізації ПЗ. Описана окремо	4,3,2,1
Оцінювання стану управління конфігурацією	Завдання виконується у складі робіт по V&V вимог до ПЗ. Перевіряється повнота і адекватність процесу управління конфігурацією СМ, описаного в документації по процесу. Що підтримує процес СМ повинен відповідати вимогам стандартів і відповідати складності, розміру і цілісності ПС, а також планам проекту і сподіванням користувачів.	4,3
Створення/ верифікація плану тестування для V&V приймання	Завдання виконується у складі робіт по V&V вимог до ПЗ. Для рівня цілісності 2 досить перевірити, що план тестування приймання, створений групою розробки ПС при виконанні основного процесу «тестування ПЗ», задовольняє потребам проекту і стандартам. Для рівнів цілісності 3 і 4 має бути розроблений самостійний план тестування приймання, підтверджуюча, що ПЗ коректно реалізує програмні і системні вимоги в середовищі експлуатації.	4,3,2
Створення/ верифікація плану тестування для V&V системи	Завдання виконується у складі робіт по V&V вимог до ПЗ. Описана окремо	4,3,2,1
Створення/ верифікація плану тестування для V&V інтеграції	Завдання виконується у складі робіт по V&V проекту ПЗ. Для рівня 2 досить перевірити, що план тестування інтеграції, створений групою розробки ПС при виконанні основного процесу «тестування ПЗ», задовольняє потребам проекту і стандартам (зокрема забезпечує відстеження інтеграції по системних вимогам). Для рівнів 3 і 4 має бути розроблений самостійний план інтеграційного тестування, підтверджуючих, що ПЗ, що створюється шляхом поступової інтеграції окремих перевірених компонентів, коректно реалізує вимоги.	4,3,2
Створення/	Завдання виконується у складі робіт по V&V проекту ПЗ.	

верифікація плану тестування для V&V компонентів	Для рівня 2 досить перевірити, що план тестування компонентів, створений групою розробки ПС при виконанні основного процесу «тестування ПЗ», задовольняє потреби проекту і використовуваним стандартам (зокрема, забезпечує відповідність компонентів до вимог до ПЗ і проекту ПЗ). Для рівнів 3 і 4 має бути розроблений самостійний план тестування компонентів, що підтверджує, що кожен компонент коректно реалізує вимоги проекту.	4,3,2
Створення/ верифікація проектів тестів для V&V	Завдання виконується у складі робіт по V&V проекту ПЗ. Для рівнів нижче 3 досить перевірити, що проекти тестів, побудовані розробниками ПС при виконанні основного процесу «тестування ПЗ» відповідають вимогам проекту та стандартам. Для рівнів 3 і 4 треба розробити проекти додаткових тестів для тестування компонентів, інтеграції системи і приймання.	
компонента		4,3,2
інтеграції		4,3,2,1
системи		4,3,2,1
приймання		4,3,2
Створення/ верифікація тестових наборів для V&V	Завдання виконується у складі робіт по V&V реалізації ПЗ. Для рівнів нижче 3 досить перевірити, що набори тестів, побудовані розробниками для тестування компонентів, інтеграції, системи і приймання ВП, відповідають відповідним проектам тестів і можуть бути прослідковані до планів тестування і системних вимог. Для рівнів 3 і 4 треба розробити набори додаткових тестів для тестування компонентів, інтеграції системи і приймання.	
компонента		4,3,2
інтеграції		4,3,2,1
системи		4,3,2,1
приймання		4,3,2
Створення/ верифікація тестових процедур для V&V	Завдання виконується у складі робіт по V&V реалізації ПЗ. Для рівнів нижче 3 досить перевірити, що тестові процедури, побудовані розробниками для тестування компонентів, інтеграції і системи, відповідають відповідним наборам тестів і можуть бути простежені по планам тестування і системним вимогам. Для рівнів 3 і 4 треба розробити додаткові тестові процедури для тестування компонентів, інтеграції і системи. Підтвердити їх відповідність планам.	
компонента		4,3,2
інтеграції		4,3,2,1
системи		4,3,2,1
Створення/ верифікація тестових процедур для V&V приймання	Завдання виконується у складі робіт по V&V випробувань ПЗ. Для рівня 2 досить перевірити, що тестові процедури приймання, побудовані розробниками, відповідають відповідним наборам тестів і можуть бути відстеженні до планів тестування і системних вимог. Для рівнів 3 і 4 треба розробити додаткові тестові процедури для тестування приймання. Підтвердження їх відповідність планам.	4,3,2
Виконання / верифікація тестів для V&V компонентів	Завдання виконується у складі робіт по V&V реалізації ПЗ. Для рівня 2 досить використовуючи результати виконання тестів компонентів розробниками, а також плани і процедури тестування компонентів підтвердити, що ПЗ задовольняє критерію приймального тестування. Для рівнів 3 і 4 треба провести тестування компонентів, проаналізувати результати і підтвердити, що ПЗ коректно реалізує проект. Підтвердити, що результати тестування простежуються до очікуваних, зафіксованих в планах. Документувати результати тестування відповідно до вимог, описаних в плані тестування, вказуючи відхилення результатів від очікуваних. Використані результати тестування підтверд-женням того, що ПЗ задовольняє установленому критерію приймального тестування.	4,3,2
Виконання / верифікація	Завдання виконується у складі робіт по V&V випробувань ПЗ. Описана окремо.	4,3,2,1

тестів для V&V інтеграції		
Виконання / верифікація тестів для V&V системи	Завдання виконується у складі робіт по V&V випробувань ПЗ. Описана окремо.	4,3,2,1
Виконання / верифікація тестів для V&V приймання	Завдання виконується у складі робіт по V&V випробувань ПЗ. Для рівня 2 досить використовуючи результати виконання тестів приймання розробниками, а також плани і процедур тестування приймання підтвердити, що ПЗ задовольняє критерію приймального тестування. Для рівнів 3 і 4 треба провести тестування приймання проаналізувати результати і підтвердити, що ПЗ задовольняє системним вимогам. Підтвердити, що результати тестування простежуються до очікуваних зафіксованим у планах. Документувати результати тестування.	4,3,2
Пробний пуск	Завдання виконується у складі робіт по V&V введення в дію ПЗ. Проводиться аналіз або виконуються тести для перевірки того, що інстальоване ПЗ відповідає тому, котрому повинно бути піддано V&V. Підтверджується, що програмний код і база даних запускаються, працюють і завершуються коректно. При переході від однієї версії ПЗ до іншої підтверджується, що ПЗ може бути демонтовано порушення функціонування решти компонентів системи.	4,3
Генерація заключного звіту про V&V	Завдання виконується у складі робіт по V&V введення в дію ПЗ. Треба підсумувати усі роботи по V&V, виконані завдання і отримані результати, включаючи місцезнаходження виявлених аномалій і їх стан. Провести оцінювання якості усього ПЗ і виробити рекомендації.	4,3,2

1.4. Управління верифікацією та валідацією критичних систем і методи перевірки

Управління діями з V&V при виконанні основних процесів ЖЦ, для перевірки результатів, здійснюється з боку організаційного процесу ЖЦ - процесу управління. У завдання процесу управління входить підготовка плану виконання будь-якого процесу (у тому числі V&V), ініціація реалізації плану, моніторинг його виконання, аналіз проблем, виявлених при виконанні плану, визначення ступеню завершеності завдань і звіт про стан керованого процесу перед керівництвом проекту.

Завдання управління верифікацією і валідацією. У завдання управління V&V входить постійний аналіз діяльності по V&V, побудова плану SVVP і його перегляд в контексті змін в планах проекту, а також координація результатів V&V з процесом розробки і іншими процесами SQA, що підтримують управління конфігурацією, спільний перегляд і аудит.

В ході управління V&V оцінюється кожна запропонована зміна в системі і ПЗ, визначаються вимоги до ПЗ, яких можуть стосуватися зміни, і планується виконання завдань V&V, пов'язаних з перевіркою внесених змін. По кожній запропонованій зміні встановлюється, чи не приведе вона до появи яких-небудь нових загроз або ризиків для ПЗ, і як воно відіб'ється на рівнях цілісності ПЗ. При зміні рівнів цілісності плани V&V

переглядаються – можуть додаватися нові завдання V&V або розширюватися сфера застосування і інтенсивність виконання раніше запланованих завдань V&V.

У контрольних точках проекту результати V&V оцінюються, і приймається рішення про перехід до наступної стадії (або множині дій) в процесі розробки. Може бути ініційоване повторне виконання V&V. Плани верифікації і валідації розробляються для всіх основних процесів ЖЦ ПС і можуть оновлюватися на вимогу замовника (споживача) (при виконанні ним процесів придбання ПС) або розробника (постачальника) (при виконанні ним процесів постачання), а також групи супроводу (при виконанні процесу супроводу).

Вихідною інформацією для побудови (оновлення) SVVP є результати виконання робіт (виходи) по процесу, що перевіряється, плани постачальника ПС, графіки проекту ПС, а також виходи виконуваного процесу V&V. Діяльність по V&V планується на підставі положень контракту між замовником і розробником і концепції ПС (базовий план SVVP).

Організувати діяльність по V&V можна по-різному. Вибір форми організації робіт в процесах V&V визначається масштабом, складністю і критичністю системи. Дії з V&V можуть виконуватися лише розробниками, рівними в правах, групою розробки (менеджерами і розробниками), групою якості, спільно представниками замовника і розробника, лише замовниками або незалежними експертами. V&V доцільно застосовувати до крупних дорогих проектів або проектів критичних по безпеці функціонування систем (safety critical system) для досягнення об'єктивності перевірок і зниження ризиків розробки. Вибираючи V&V, як форму організації V&V проекту, замовник покладається на орган V&V як на об'єктивну і незацікавлену сторону для здобуття інформації про дійсний стан розробки, дотримання термінів проекту і витрат на його виконання.

Технічна незалежність групи V&V полягає в тому, що її члени не приймають участі в розробці ПС, проте володіють знаннями і досвідом розробки систем даного класу і компетентні в ухваленні рішень, що стосуються технічних аспектів ПС. Вони не схильні до впливу або тиску з боку групи розробки при виконанні V&V. Технічна незалежність забезпечує можливість ретельно вивчити тонкощі прийнятих розробниками рішень і виявити помилки, що залишилися ними непоміченими. Для виконання робіт група може використовувати ті ж інструментальні засоби підтримки розробки, що і розробники, проте заздалегідь потрібно провести випробування цих засобів, аби гарантувати, що вони не маскують помилок в аналізованому і протестованому ПЗ. По можливості, група V&V повинна застосовувати або розробляти свій набір інструментів аналізу і тестування.

Адміністративна незалежність означає, що V&V виконується організацією, адміністративно не пов'язаною ні з розробником, ні із замовником. Орган V&V має право на свій розсуд вибирати ті частини системи, для яких проводить V&V, визначати аспекти і методи перевірки і встановлювати графік робіт по V&V. Результати перевірки доводяться як

до замовника, так і до розробника ПС. Координувати роботи органу V&V із зацікавленими організаціями може третя сторона – організація-координатор.

Структура і вміст плану верифікації і валідації. У плані SVVP фіксується інформація, необхідна для управління і виконання процесів, дій і завдань по V&V в ході ЖЦ ПС. Зразкова структура плану показана на рисунку 1.6. Ця структура необов'язкова і може бути змінена за умови збереження вмісту плану. Роз'яснення по складанню плану SVVP містяться в стандарті IEEE Std. 1012. Якщо по якомусь з пунктів плану інформація не надається, це повинно вказуватися фразою «дана тема не охоплюється планом» і обґрунтовуватися. У план можуть бути додані і інші пункти.

План верифікації і валідації ПС

1. Призначення
2. Посилання
3. Визначення
4. Організація роботи по V&V
   - 4.1. Організація
   - 4.2. Графік робіт
   - 4.2 Схема призначення рівнів цілісності ПО
   - 4.4. Розподіл ресурсів
   - 4.5Відповідальності
   - 4.6. Інструменти, методології і методи
5. Процеси і дії з V&V
   - 5.1. Процес: Управління
     - 5.1.1. Дія: Управління V&V
   - 5.2. Процес: Придбання
     - 5.2.1. Дія: V&V підтримки придбання
   - 5.3. Процес: Постачання
     - 5.3.1. Дія: V&V планування
   - 5.4. Процес: Розробки
     - 5.4.1. Дія: V&V Концепції
     - 5.4.2. Дія: V&V Вимог
     - 5.4.3. Дія: V&V Проекту
     - 5.4.4. Дія: V&V Реалізації
     - 5.4.5. Дія: V&V Випробувань
     - 5.4.6. Дія: V&V Введення в дію
   - 5.5. Процес: Експлуатація
     - 5.5.1. Дія: V&V Експлуатації
   - 5.6. Процес: Супровід
     - 5.6.1. Дія: V&V Супроводу
6. Вимоги до звітних документів V&V
7. Вимоги до адміністративних процедур
   - 7.1.Реєстрація і ухвалення рішень по аномаліях
   - 7.2. Політика ітерації завдань
   - 7.3. Регулювання відхилень від плану
   - 7.4. Процедури контролю
   - 7.5. Стандарти, практичне керівництво і угоди
8. Вимоги до документування V&V

Рис. 1.4. Структура плану SVVP

1.5. Види і методи перевірки програмних систем. Огляд аналітичних методів

Методи, вживані при виконанні процесів верифікації, валідації, спільного перегляду і аудиту, можна класифікувати по-різному, наприклад:

- динамічні (пов'язані з виконанням програм) і статичні (пов'язані з переглядом тексту);
- пошукові (направлені на пошук помилок) і такі, що підтримують ухвалення рішень (вживані для аналізу робочих продуктів з метою виявлення яких-небудь відмітних особливостей, виконання оцінок і ін.);
- для колективної роботи і для індивідуальної (аналітичною) роботи;
- формальні (котрі вимагають чіткого дотримання методу) і неформальні;
- колегіальні (виконувані колегами по роботі, рівними в правах, і що не загрожують адміністративними наслідками) і ревізійні і тому подібне.

Одні і ті ж методи можуть застосовуватися для виконання різних процесів перевірки, і, навпаки, при виконанні одного процесу перевірки може використовуватися декілька методів. Часто метод асоціюється із завданням процесу, для вирішення якої він застосовується, а саме завдання має назву, однойменне методу, наприклад, аналіз відстеження, аналіз критичності і ін. Щоб уникнути плутанини з однойменними найменуваннями методів, процесів і завдань далі при описі методів наводяться їх еквівалентні назви англійською мовою. Для зручності викладу ми використовуємо класифікацію методів на колективні та індивідуальні аналітичні і попутно характеризуємо їх з точки зору інших класифікаційних ознак. Розглядаються лише методи статичної перевірки.

Аналітичні методи призначені для індивідуального дослідження робочих продуктів ПС з використанням (або без) інструментальних засобів підтримки. Вони можуть бути віднесені до категорії пошукових методів, методів підтримки ухвалення рішень або мати подвійне призначення. У таблиці 1.7 дана коротка характеристика методів з вказівкою посилань на доступні джерела з їх описом. Потрібно відмітити, що ряд аналітичних методів індивідуального вживання, які можуть використовуватися для цілей V&V, мають набагато ширше поле додатка (наприклад, в процесах SQA, аналізу вимог, проектування, тестування або управління ризиком).

Таблиця 1.7. Методи аналізу робочих продуктів ПС

Метод	**Коротка характеристика і рекомендації по застосуванню**
Перелік питань (Checklists)	Перелік питань, що стосуються певного аспекту перевірки робочого продукту сстворюється для кожного виду робочих продуктів (вимог, проекту та ін.) з урахуванням особливостей класу ПС (ПС реального часу, інформаційні та ін.). Структурований по категоріях можливих дефектів. Використовується один лист з запитаннями для кожної категорії. Може використовуватися для пошуку дефектів і обґрунтування рішень, що приймаються.
Аналіз алгоритму	Мета – визначити функціональну коректність алгоритму і його

(Algorithm analysis)	операційні характеристики. Припускає повторне виведення рівнянь або оцінку придатності чисельних методів. Алгоритми аналізуються з точки зору їх коректності, ефективності, простоти, оптимальності і точності. Досліджує вплив округлень, оцінює забезпечувану точність зберігання при використанні різних типів перемінних (наприклад, з простою або подвоєною точністю) і вплив перетворення типів. Виявляє неправильні, неефективні (по пам'яті і часу) або нестійкі алгоритми; недостатню точність обчислювань; непрацездатність на повному діапазоні даних; неправильний аналіз погрішності обчислення та ін. Може використовуватися для пошуку дефектів.
Аналіз інтерфейсу (Interface analysis)	Перевірка інтерфейсів різних типів - зовнішнього, внутрішнього, апаратного, програмного, програмно-апаратного і програмно-інформаційного (бази даних). Може включати наступне: - аналіз ситуації, коли усі змінні інтерфейсу мають критичні значення; - аналіз ситуації, коли частина змінних інтерфейсу має критичні значення, а інші - мають нормальні значення; - аналіз ситуації, коли одна (по черзі кожна) зміна інтерфейсу набуває усіх значень з області значень, а інші - тільки нормальні значення. Виявляє наступні типи помилок : помилки опису входів/виходів; невідповідність фактичних і формальних параметрів (точність, тип, кількість); некоректне використання функцій чи виклик підпрограм; неузгодженість атрибутів глобальних змінних та ін. Може використовуватися для пошуку дефектів.
Аналіз потоків даних (Data flow analysis)	Застосовується для виявлення невизначених входів/виходів чи форматів даних, порушення порядку обробки (читання даних до їх запису), відсутності обробки (неодноразового запису в одні і ті ж змінні без використання (читання) записаних даних). Використовує інформацію про потоки управління і про структури зберігання(змінних), в які пишуться або з яких читаються дані, що обробляються додатком. Аналіз потоку інформації – розширення аналізу потоків даних автоматизований в сучасних CASE-інструментах. Може використовуватися для пошуку дефектів і обґрунтування рішень, що приймаються.
Аналіз потоку інформації (Information flowanalysis)	Використовується для визначення об'єму і складності внесення змін в робочі продукти на певній стадії ЖЦ. Основні стадії - огляд запиту про зміни, оцінка міри розповсюдження змін, оцінка витрат і ресурсів, аналіз користі/витрат. Використовує аналіз трасованості, аналіз взаємозалежності елементів системи, аналіз узгодженості. Може використовуватися для обґрунтування рішень, що приймаються.
Аналіз операційного профілю (Operational profile analysis)	Профіль – повна множина альтернатив (наприклад, множина альтернативних сценаріїв роботи користувачів), для кожної з яких існує певна *вірогідність* появи. Повнота множина альтернатив означає, що сума вірогідності їх появи дорівнює одиниці. Аналіз операційного профілю використовується для раціонального розподілу зусиль із V&V робочих продуктів ПС, побудови системи тестів і тестових сценаріїв, максимально наближених до реальних умов середовища функціонування ПС та ін. Може використовуватися для обґрунтування рішень, що приймаються.
Аналіз трасованості (Traceability analysis)	Існує декілька типів аналізу трасованості – трасування вимог, трасування проекту, трасування коду і трасування тесту. Перевіряється, що кожна вимога знайшла відображення в

	проекті/коді, що усі аспекти проекту/коду ґрунтовані на визначених вимогах, що тестування дає результати, сумісні з установленими вимогами. Ґрунтований на застосуванні матриць трасування вимог: - матриця «вимоги/*метод оцінювання»* – для перевірки охоплення усіх вимог певною формою V&V; - матриця «вимоги/тести» – для перевірки охоплення усіх вимог тестуванням; - матриця «вимоги/модулі» - для перевірки покриття кожної вимоги групою модулів (програмних компонентів), що реалізують цю вимогу. Типи виявлених помилок : - *при аналізі трасованості вимог* – пропущені функції, неправильне впорядкування вимог по важливості, специфікація ПЗ несумісна з іншими специфікаціями системи; - *при аналізі трасованості проекту* – нереалізовані вимоги, неправильна інтерпретація специфікованих вимог детальний проект не відповідає архітектурному; - *при аналізі трасованості коду* – невідповідність коду проекту невідповідність коду стандартам, ПЗ в цілому або окремі компоненти не виконують необхідні функції та ін. Може використовуватися для пошуку дефектів і обґрунтування рішень, що приймаються.
Аналіз дерев подій (Event tree analysis) (ETA)	Ідентифікує компоненти, події відмов яких можуть впливати на безпеку функціонування системи. *Наслідки* кожної події (що викликає низку інших подій) трасуються до тих пір, поки не проясняться небезпеки (самого верхнього рівня) пов'язані з цими подіями. По ходу ідентифікації хронологічного ланцюжка подій визначається вірогідність кожної події і комбінації подій. Дерево подій представляє повний спектр можливих *наслідків* певної події відмови компонента (що являється коренем дерева). Не відбиває *причин* появи події відмови. Може використовуватися для пошуку дефектів.
Аналіз дерев відмов (помилок) (Fault tree analysis) (FTA)	Можлива відмова компонента системи розглядається як подія верхнього рівня, *причини* якого приховані усередині компонента. Аналіз дерева відмов - це процес *ідентифікації ризиків*, зв'язаних з компонентом системи. Дерево відмов – ієрархічна структура, на вершині (в корені) якої - стан відмови компонента, прилеглі гілки – події, що призводять до переходу в цей стан. Витік кожної гілки – стан елементу компонента що привело до настання відповідної події. Це *дедуктивний* метод пошуку умов чинників (причин) появи події відмови компонента. Трасування подій може розпочинатися не з події відмови компонента системи, а із зовнішньої події, що несе загрозу. Може використовуватися для пошуку дефектів.
Аналіз режимів (механізмів) і наслідків відмови (Failure mode and effect analysis (FMEA))	Методи *відвертання появи відмов*. Використовуються для ідентифікації типів потенційних дефектів і механізмів їх появи (режимів роботи системи, в яких вони можуть з'явитися) з метою визначення їх можливого впливу на об'єкт, що вивчається (систему, компонент), а також класифікації типів дефектів по критичності наслідків (у критичних системах). Широко застосовуються в інженерії надійності і безпеки технічних систем. Представляють підхід, зворотний (індуктивний) по відношенню до методу FTA (напрям аналізу – від події відмови окремого компонента (з урахуванням його критичності або без) до небезпечного зовнішньої події для користувача). Ідентифікують ті компоненти, які вимагають підвищеного уваги при розробці. Корисні при визначенні стратегії управління ризиком, тестування ПЗ в умовах обмежених ресурсів, забезпечення відмово-стійкості ПЗ, побудови процедур супроводуу та ін. Можуть

	використовуватися для обґрунтування рішень, що приймаються.
Аналіз режимів і критичності відмов (Failure mode and criticality analysis (FMECA))	Небезпека/загроза – стан системи або середовища, потрапляння до якого призводить до катастрофічних наслідків для суспільства. У програмній інженерії термін «небезпека» частіше зв'язується з процедурами забезпечення *безпеки функціонування* (safety), а «загроза» – з процедурами *захисту інформації (*security).Методи застосовуються при розробці високо-цілісних систем. Використовують метод ETA. Можуть використовуватися для пошуку дефектів і обґрунтування рішень, що приймаються.
Аналіз загроз (Threat analysis)	Небезпека/загроза – стан системи або середовища, попадання в яке призводить до катастрофічних наслідків для суспільства. У програмній інженерії термін «небезпека» частіше зв'язується з процедурами забезпечення *безпеки функціонування* (safety), а «загроза» – з процедурами *захисту інформації(*security). Методи застосовуються при розробці високо-цілісних систем. Використовують метод ETA. Можуть використовуватися для пошуку дефектів і обґрунтування рішень, що приймаються.
Аналіз критичності (Criticality analysis)	Використовується для визначення стратегій V&V, застосовних до компонентам (модулям) з високим ризиком відмов. Ранжує компоненти по мірі вкладу у безпеку. Встановлює, в якій мірі безпека функціонування системи залежить від коректної роботи окремих компонентів ПЗ, і яким чином на неї можуть вплинути відхилення в реалізації ПЗ. Класифікація компонентів за рівнем критичності може виконуватися за допомогою вказівки їх ролі в забезпеченні безпеки (не грає, грає побічно, грає безпосередньо, грає допоміжну роль), а також за допомогою індексу критичності. Використовує результати аналізу небезпеки, FMEA, FMECA для ідентифікації вимог(елементів проекту, коду), неправильна реалізація яких може викликати найбільш серйозні наслідки. Може використовуватися для обґрунтування рішень, що приймаються.
Ортогональна класифікація дефектів (ODC) (Orthogonal defect classifi-cation)	Характеристика кожного виявленого дефекту по трьом аспектам: – *тип дефекту*. Призначається виходячи з характеру дій, які мають бути виконані для усунення дефекту (виправлення документації, інтерфейсів, функцій, алгоритмів, призначень значень та ін.). Служить мірою досконалості продукту; - *тригер дефекту (*причина виявлення). Напрям перевірки (у рамках одного з трьох видів перевірки – огляди, автономне і функціональне тестування, системне тестування і випробування), при виконанні якої був виявлений дефект (перевірка повноти, перевірка узгодженості, нормальний режим та ін.). Служить мірою ефективності V&V; – *вплив дефекту*. Критерії оцінки дії, яка дефект може зробити (чи вже зробив) на міру задоволеності кінцевого користувача продуктом (функціональність, надійність, зручність застосування та ін.). Служить мірою якості продукту при використанні. ODC - парадигма виміру характеристик процесів і продуктів на основі причинно-наслідкового аналізу дефектів. Може використовуватися для обґрунтування рішень, що приймаються.
Аналіз складнощі (Complexity analysis)	Використовується для виявлення компонентів проекту або коду складних для коректної реалізації, тестування і супроводу. Може використовуватися для обґрунтування рішень, що приймаються.
Розробка прототипу (Prototyping)	Застосовується для узгодження і оцінки тієї, що реалізовується вимоги, оцінки проекту інтерфейсу користувача, пророцтва розміру і складнощі ПС, визначення стратегій тестування. Може використовуватися для пошуку дефектів.

Розділ 2. Тестування та побудова стратегії тестування критичних програмних систем

2.1. Основні поняття тестування

Переходимо до тестування, як основного засобу перевірки і виявлення дефектів з метою забезпечення надійної безпеки критичних систем.

Огляд потоку артефактів зображено на рисунку 2.1.

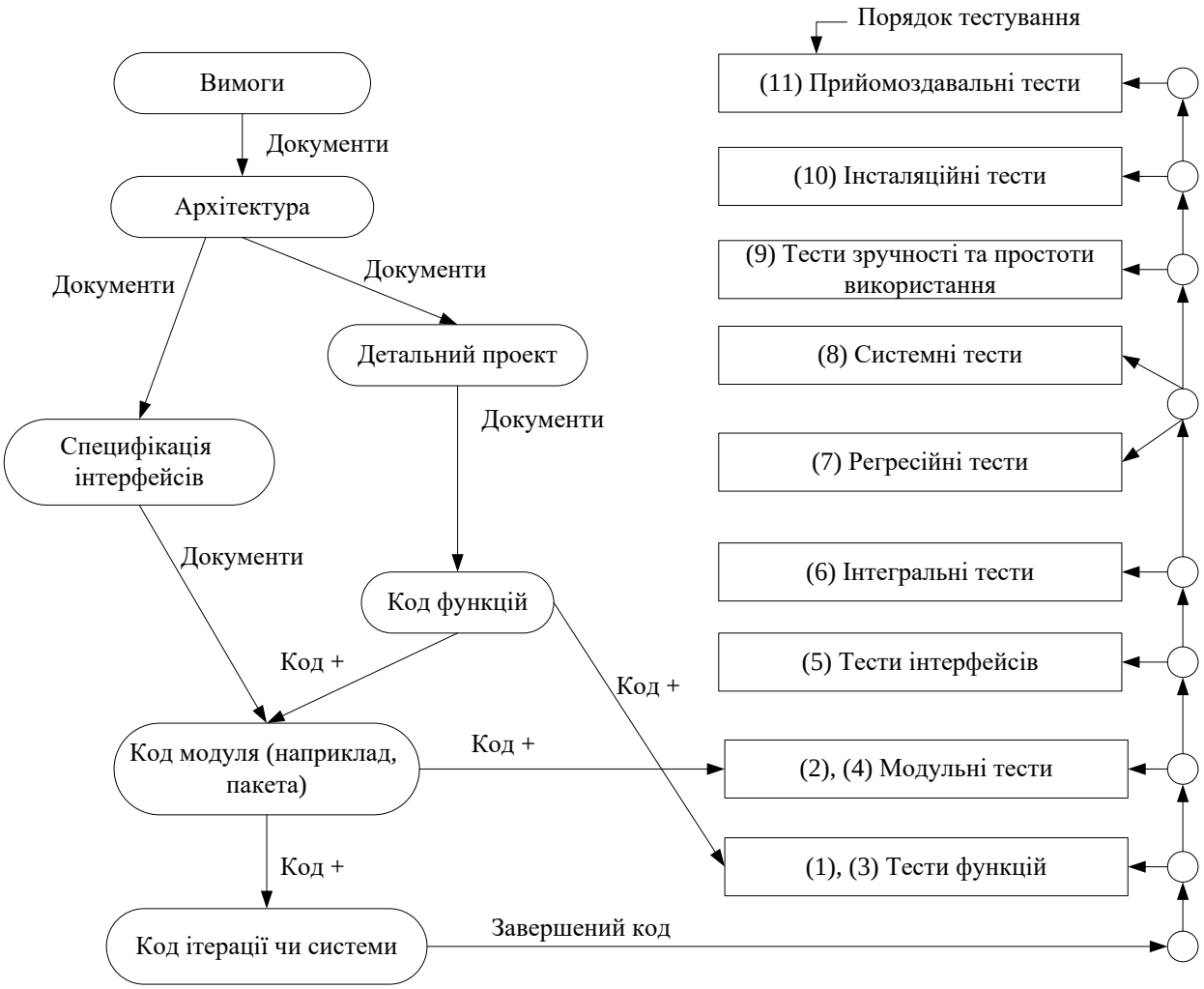

Рис. 2.1. Огляд тестування: потік артефактів

Тестування – невід'ємна складова процесу програмної інженерії, один з методів подальшого поліпшення якості розробленого програмного забезпечення системи за допомогою виявлення дефектів, що залишилися, не виявлених раніше всіма іншими видами перевірок.

Термін testing (тестування) широко використовується в літературі, але визначається по-різному. Стандарт ANSI/IEEE Std. 610.12:1990 визначає термін testing в самому його широкому сенсі як будь-яке ді. з аналізу ПР (включаючи методи як динамічної, так і статичної перевірки). Слово testing може бути переведене не лише як тестування, але і як випробування (як це зроблено в стандартах ДСТУ 2844-94. ДСТУ 2853-94. Проте, поняття «випробування» нам звичніше пов'язувати з тими, що завершують стадії ЖЦ, на яких виконується не пошук помилок, а підтвердження придатності розробленого програмного продукту.

У данному розділі тестування розглядається як процес виконання програмної системи (або елементів ПС) з метою перевірки її відповідності встановленим вимогам і виявлення дефектів. У міру розвитку програмної інженерії розуміння цілей і завдань тестування змінювалося. Виділяють наступні «історичні» періоди, що відображають прогрес в розумінні цілей тестування і його місця в життєвому циклі ПС:

1) період до 1956 року – орієнтація на налагодження;

2) період з 1957 по 1978 років – орієнтація на встановлення відповідності ПС початковим вимогам;

3) період з 1979 по 1982 років – орієнтація на виявлення дефектів, що залишилися після реалізації;

4) період з 1983 по 1987 років – орієнтація на аналіз, перевірку і тестування з метою оцінки якості ПС на всіх стадіях розробки;

5) період з 1988 по 1995 роки – інтеграція дій з перевірки і тестування в життєвий цикл ПС з метою запобігання появи дефектів на всіх стадіях розробки (з охопленням всіх дій з верифікації, валідації і тестування).

З появою в 1995 році стандарту ISO/IEC 12207, в якому всі дії, пов'язані із створенням ПС, представлені у вигляді окремих процесів ЖЦ, сталося розділення завдань верифікації, валідації і тестування по окремим процесам. Тестування віднесене до основних процесів, але представлене не одним, а групою процесів. Крім того, ряд завдань тестування вирішуються в рамках інших процесів розробки, зокрема, завдання планування тестування розподілені по процесах: «Проектування архітектури системи», «Аналіз вимог до ПЗ», «Проектування ПЗ». Автономне і інтеграційне тестування ПЗ виконуються в рамках процесів «Побудова ПЗ» і «Інтеграція ПЗ», оскільки нерозривно з ними пов'язані.

Таким чином, сталася інтеграція тестування з процесами розробки, і сьогодні тестування розглядається як безперервна діяльність, що виконується впродовж всього процесу розробки. Планування тестування повинне починатися на стадії аналізу вимог, а плани і процедури тестування повинні систематично і постійно уточнюватися по мірі розробки системи. У SWEBOK галузь знань «Тестування ПЗ» представлена п'ятьма основними розділами (рисунок 2.2).

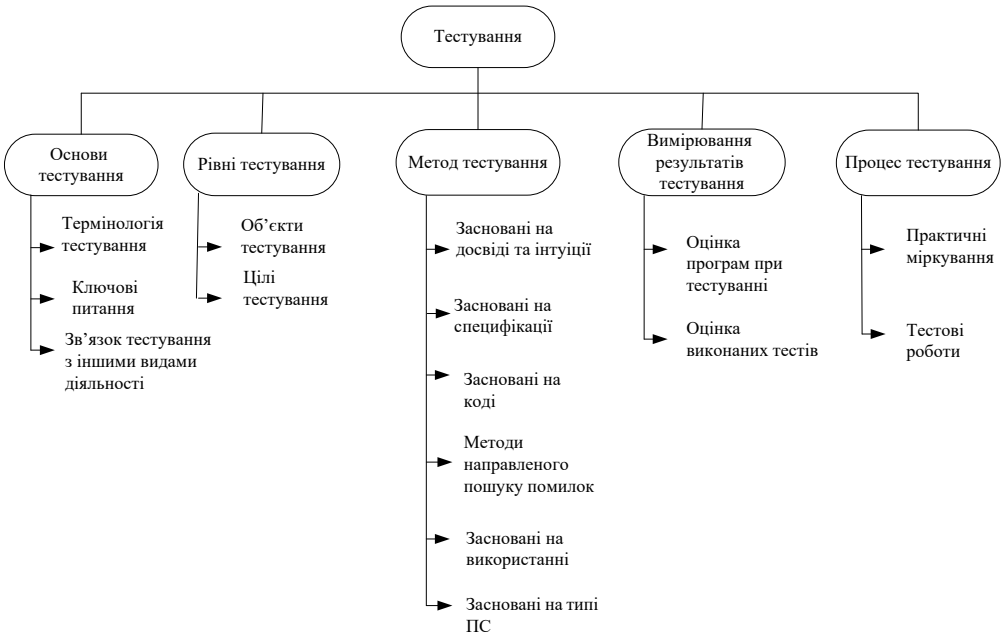

Рис. 2.2. Галузь знань «Тестування ПЗ» в SWEBOK

Термінологія тестування. Тестування полягає в динамічній перевірці поведінки програми на кінцевій множині тестових даних, спеціальним чином вибраних з нескінченого вхідного простору, на відповідність встановленій очікуваній поведінці.

Виділені курсивом слова ключові для тестування:

- динамічне: тестування завжди передбачає виконання програми;
- кінцеве: навіть для невеликої програми теоретично можливо створити таку кількість тестів, для виконання яких можуть знадобитися роки.

Неповнота – це одна з основних проблем тестування, оскільки на практиці множину тестів можна розглядати як нескінченне. Кількість же тестів, які можуть бути виконані в обмежені терміни, є певним конкретним значенням. Таким чином, тестування завжди передбачає деякий «компроміс» між обмеженими термінами і потенційно необмеженою кількістю тестів. Це призводить до відомих проблем тестування, таким як ухвалення рішень про адекватність тестування, і проблемам управління – оцінці витрат (вартості, часу, персоналу) на тестування;

- вибраних: з проблемою адекватності тестування пов'язана проблема вибору обмеженої множини тестів. Методи тестування, в основному, відрізняються підходами до вибору множини тестових даних та ТНД з вхідного простору;

- очікувана поведінка: потрібно уміти визначати, чи правильні отримані результати виконання програми, чи відповідає спостережуване виконання очікуванням користувача або специфікаціям. У літературі по тестуванню це називається проблемою оракула (еталону), для вирішення якої можуть приміняться різні підходи (оцінка результатів, порівняння з існуючим еталоном, узгодження з користувачем трактувань понять «дефекту» і «відмова»).

Хоча основна мета тестування - виявити дефекти в програмній системі і встановити її функціональну придатність, необхідно розглядати і інші його цілі - тестування зручності вживання, продуктивності та інше. Звідси слідують два основні підходи до виконання тестування: деструктивний (негативне, руйнівне тестування) і конструктивний (позитивне або демонстраційне).

При деструктивному підході тести вибираються з метою виявлення дефектів, і ефективним вважається тест з високою вірогідністю їх виявлення.

При конструктивному підході тести вибираються для демонстрації відповідності характеристик ПС встановленим вимогам користувача або цілям якості.

Тестування програмної системи впродовж процесу розробки виконується на декількох рівнях. Для кожного рівня тестування визначається категорія об'єктів тестування (вся система, програмні компоненти, модулі) і набір цілей, що перевіряються (трестованих характеристик).

Цілі і об'єкти спільно визначають критерій вибору множини тестів, як відносно його повноти – «який об'єм тестування достатній для досягнення встановленої мети?», так і його складу – «які тести мають бути вибрані для досягнення встановленої мети?». Перше питання стосується вибору критерію покриття, а другий - критерію вибору типів тестів. На кожному рівні тестування повторюється багато разів, утворюючи цикли тестування-виправлення-повторне тестування.

У сучасній програмній інженерії всі види дій, пов'язані з тестуванням, починаючи з планування до оцінки результатів, повинні інтегруватися в чітко визначений, документований і контрольований процес тестування. Документування процесу тестування полегшує взаємодію між розробниками, групою тестування і керівництвом проекту, а також дозволяє зробити процес видимим, повторюваним і вимірюваним.

До ключових питань тестування в SWEBOK віднесені наступні:

- Критерії вибору тестів/Критерії адекватності тестів (або правила припинення тестування). Ці критерії можуть застосовуватися як для створення набору тестів, так і для перевірки, наскільки вибрані тести адекватні завданням (тестування), а також допомагають визначити, коли можна або необхідно припинити тестування.

- Ефективність тестування/Мети тестування. Тестування – це спостереження за поведінкою програми, що виконується в цілях тестування із заданими параметрами, по заданому сценарію або з іншими заданими початковими умовами або цілями тестування. Ефективність тесту може бути визначена лише в контексті заданих умов.

- Тестування для виявлення дефектів. У тестуванні для виявлення дефектів застосовується деструктивний підхід; а успішним вважається тест, що виявив дефект. Цей підхід принципово відрізняється від іншого підходу, коли тести запускаються для демонстрації того, що програма задовольняє пред'явлені до неї вимоги і, відповідно, тест вважається успішним, якщо не знайдено дефектів.

- Проблема оракула. «Оракул» в тестуванні – це будь-який агент (людина або програма), що оцінює поведінку програми і що формує висновок про результат тесту (тест пройдений чи ні). Цей висновок істотно залежить від трактування поняття «відмови» і «дефект» в конкретному контексті.

- Теоретичні і практичні обмеження тестування. Обмеження зумовлені неможливість вичерпного тестування і його економічної недоцільності для конкретних ПС. Тестування повинно проводитися з урахуванням ризику відмови ПС і ґрунтуватися на таких стратегіях тестування, які отримали поширення в сучасних методологіях розробки, як тестування, засноване на ризику (risk-based testing), і кероване ризиком тестування (risk-driven testing), які коротко розглянуті в цьому розділі.

- Проблема нездійсненних шляхів. Нездійсненні шляхи – це шляхи потоку управління програми, які не можуть бути виконані ні при яких вхідних параметрах. Це складна проблема шляхового тестування і особливо його автоматизації.

- Тесто-придатність. Цей термін має два різні значення. Перше – міра легкості опису критеріїв тестового покриття для ПС. Друге – вірогідність, що виконання програми при тестуванні призведе до її відмови, за наявності дефекту.

Зв'язок тестування з іншими видами діяльності. Тестування має багато загального з процесами верифікації і валідації (V&V). Спільність процесу тестування з процесами V&V полягає в цілісності складу і структури планів, що рекомендуються IEEE Std 829, а також об'єктів і вживаних методів. Відмінність цих процесів полягає в умовах їх застосування. Тестування – основний процес в ЖЦ, що виконується завжди, для всіх об'єктів ПЗ системи незалежно від її критичності. Процеси V&V, в сучасному трактуванні стандартів IEEE Std. 1012 і ISO/IEC 12207 – підтримуючі процеси, які можуть застосовуватися до вибраних об'єктів тестування для перевірки планів їх тестування і підтвердження того, що виконане тестування адекватно рівню критичності ПС. По відношенню до процесу тестування V&V виконує контрольну функцію і підтверджує відповідність об’єктів встановленим вимогам.

Тестування ПС тісно пов'язане з налагодженням і власне програмуванням, але охоплює набагато ширше коло проблем і учасників – програмістів, тестувальників, аналітиків і інженерів по якості.

2.2. Види та рівні тестування

Структуризація процесу тестування ПС по рівнях зазвичай зв’язуються або з видами тестованих об'єктів (модуль, система або її частини), або з цілями якості ПС (функціональність, безпека, надійність і ін.), що перевіряються, на різних стадіях її створення і експлуатації.

2.2.1. Рівні тестування по видах об'єктів

Традиційно виділяється три рівні тестування ПЗ: автономне або модульне (unit testing), інтеграційне (integrating testing) і системне (system testing). У стандарті ISO/IEC 12207 сформульовано чотири рівні тестування:

- модульне (у процесі «Побудова (Конструювання) ПЗ»);
- інтеграційне (у процесі «Інтеграція ПЗ»);
- тестування ПЗ (як процес);
- системне (у процесі «Системна інтеграція»).

Модульне тестування передбачає перевірку функціонування об’єктів в ізоляції один від одного. Воно зазвичай виконується розробниками з доступом до коду і чергується з налагодженням. Об'єктами тестування є окремі процедури (методи і класи при об'єктному підході), програмні модулі і програмні компоненти, що складаються з тісно зв'язаних модулів.

Цілі модульного тестування - виявлення дефектів в реалізації функцій об'єктів і підтвердження відповідності об'єкту специфікаціям проекту (технічному проекту).

Якнайповніше питання систематичного модульного тестування висвітлені в стандарті IEEE 1008-87 "Standard for Software Unit Testing".

Інтеграційне тестування призначене для перевірки правильності взаємодії між програмними об'єктами, протестованими автономно. Сучасні систематичні стратегії інтеграції визначаються архітектурою ПС і моделлю розробки (зазвичай ітераційною з приростами). Інтеграційне тестування виконується при кожній збірці нової версії ПС з метою виявлення дефектів в інтерфейсах між інтегрованими компонентами і підтвердження їх відповідності проекту архітектури ПС.

Тестування програмного забезпечення полягає в перевірці функціонування інтегрованої версії ПЗ системи в модельованому середовищі. Цілі тестування на цьому рівні - виявлення дефектів в реалізації зовнішніх функцій ПЗ і підтвердження його відповідності специфікації функціональних вимог.

Виділення рівня системного тестування зумовлене необхідністю забезпечення і оцінки якості сполучення ПЗ з іншими, у тому числі, не програмними компонентами сучасних систем. На цьому рівні тестування відбувається перевірка відповідності ПС цілям якості, встановленим у вимогах до системи (з акцентом на не функціональні вимоги), таким як надійність, стійкість, продуктивність, переносимість і ін., а також зовнішнім інтерфейсам з іншими системами, середовищем, апаратним забезпеченням. Велика частина функціональних відмов і дефектів має бути ідентифікована і усунена на попередніх рівнях тестування. Ціллю тестування на даному рівні – виявлення дефектів, пов'язаних з незадовільними технічними характеристиками функціонування системи, і підтвердження її відповідності проекту архітектури системи.

Взаємозв'язок рівнів і цілей тестування можна представити у вигляді модифікованої каскадної моделі ЖЦ (так званої моделі V-подібної), що відображає процеси декомпозиції та інтеграції ПС і відповідні рівні тестування (рисунок 2.3).

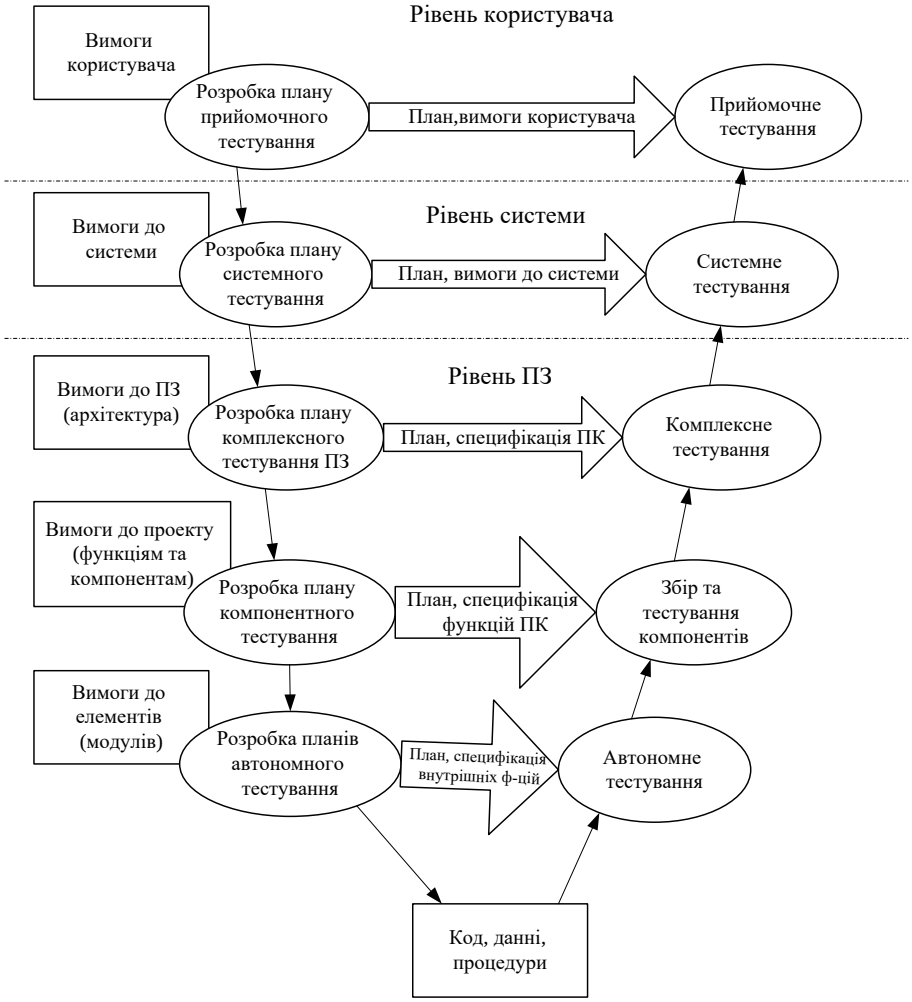

Рис. 2.3. V – подібна модель тестування

У цій моделі тестування розглядається як безперервний процес, інтегрований в процес розробки ПС і що включає два взаємозв'язаних підпроцеси – планування тестування в рамках

процесів розробки системи (ліва гілка на рисунку) і проведення тестування відповідних об'єктів (права гілка).

Хоча на рисунку для наочності рівні тестування відображують в прив’язці до традиційної каскадної моделі розробки, – на практиці залежно від архітектури системи, а також прийнятої моделі розробки, ці рівні можуть об'єднуватися, а різні завдання тестування –- виконуватися паралельно.

На практиці завдання тестування ПЗ і системного тестування часто об'єднуються в єдиний процес, але для складних ПС тестування технічних характеристик повинне виконуватися окремо.

2.2.2. Види випробувань програмної системи

Окрім розглянутих вище рівнів тестування об'єктів ПС, існують види тестування ПС в цілому, виконувані на завершальних стадіях її розробки і в ході експлуатації. До них, перш за все, відносяться випробування ПС. У ДСТУ 2853-94 визначені наступні види випробувань: попередні, приймальні, встановлювальні і експлуатаційні.

Попереднє випробування програмної системи можна вважати найвищим рівнем «формального» тестування, що виконувалося в середовищі розробки з метою виявлення можливих дефектів, що залишилися, і оцінки досягнутого рівня якості ПС. Тут «формальний» означає, що тестування виконується суворо відповідно до встановлених документованими процедурами і за участю представників замовника. Випробування здійснюються в два прийоми – тестування ПЗ і тестування системи. Отже, мета попередніх випробувань – це виявлення невідповідності ПС зовнішнім специфікаціям функціональних і технічних характеристик.

Приймальне і всі подальші випробування безпосередньо не зв'язуються з поняттям тестування, бо їх мета – підтвердити відповідність (або невідповідність) системи початковим вимогам користувача. Відповідно до стандарту ISO/IEC 12207 ці випробування виконуються в рамках різних процесів ЖЦ.

Мета приймальних випробувань (приймального, кваліфікаційного тестування) – визначення міри відповідності розробленої ПС початковим вимогам замовника (користувача). Ці випробування проводяться виключно в контексті вимог замовника (користувача) і з його безпосередньою участю і, як правило, в середовищі експлуатації. У трактуванні стандарту ISO/IEC 12207 приймальне тестування, як вид тестування, не включено в «Процес тестування», а виконується в рамках процесів «Постачання» і «Приймання замовником» (користувачем) і зв'язується з процесом «Валідації».

Тестування інсталяції (випробування встановлень) виконується в середовищі експлуатації (в рамках процесу «Інсталяція ПС») і відноситься до рівня системного тестування. Включає тестування процедур інсталяції ПС з зовнішніх носіїв.

Перед остаточним випуском системи проводиться Альфа і Бета тестування. Для цього система передається невеликій групі користувачів – внутрішніх (Альфа) або зовнішніх (Бета), які експлуатують систему і інформують розробника про виявлені проблеми. Виявлені відмови і дефекти свідчать про якість виконаного раніше тестування.

Регресійне (regression test) і повторне (re-test) тестування. Ці поняття в літературі часто змішуються, оскільки обидва види тестування стосуються повторного виконання тестів. Проте вони мають деякі відмінності. Повторне тестування виконується на будь-якому рівні тестування для перевірки внесених змін, і не регламентується планом тестування. Регресійне тестування пов'язане з розвитком ПС і особливо широко застосовується в ітераційних моделях розробки (для тестування нових версій ПС). Воно полягає в повторенні підмножини раніше виконаних тестів (для того, щоб переконатися в тому, що те, що раніше працювало, ще працює), а також розробці нових тестів для перевірки правильності внесених змін. На відміну від повторного тестування, регресійне тестування планується. При його плануванні вирішується проблема відбору мінімального набору регресійних тестів.

2.2.3. Види тестування характеристик програмної системи

По відношенню до характеристик, що перевіряються, і цілей якості будь-якого об'єкту системи можна виділити наступні види тестування.

Функціональне тестування (тестування на відповідність або тестуванні коректності) направлене на перевірку відповідності спостережуваної поведінки системи (або окремих елементів) специфікаціям (зовнішнім і внутрішнім). Виконується, як правило, в середовищі розробки по планах, розроблених на основі специфікацій функціональних вимог. Мета функціонального тестування полягає у виявленні невідповідностей між реальною поведінкою реалізованих функцій і початковими вимогами, представленими в специфікації функцій, і визначенні повноти функціонального покриття відносно специфікації функцій. Виконується на рівнях модульного тестування і тестування ПЗ в цілому. Функціональні тести можуть бути отримані по зовнішніх специфікаціях функцій, проектною інформації або прототипу ПЗ.

Тестування безпеки (Security testing) полягає в перевірці можливості несанкціонованого доступу до ПС. Виконується шляхом моделювання можливих атак зовнішніх користувачів (у тому числі інших ПС) з метою «зламати» систему захисту.

Тестування зручності вживання (ергономічності) призначене для оцінювання простоти вживання і вивчення системи кінцевими користувачами, ефективності її використання для

вирішення завдань користувача і здатності до відновлення при помилках користувача. Часто включає також тестування документації (on-line, довідкової служби, підсистеми навчання).

Цей вигляд тестування вимагає знання сучасних стандартів по ергономіці, наприклад, ISO 13407:1999, а також погоджених із замовником вимог до призначеного для користувача інтерфейсу. Застосовується в рамках процесу аналізу вимог (тестування прототипу), тестування програмних компонентів, що реалізують інтерфейс, тестування ПЗ і приймальних випробувань системи.

Тестування технічних характеристик. Це процес пошуку невідповідностей програмної системи цілям якості (цільовим вимогам), зафіксованим в зовнішніх специфікаціях разом з описом її функцій. Відправною точкою для його виконання є вимірність і принципова досяжність встановлених цільових вимог (до надійності, продуктивності, сумісності, конфігурації і ін.). Ці види тестування отримали назви, відповідні цілям тестування, і зазвичай проводяться в рамках тестування системи в спеціально модельованому середовищі, максимально наближеному до середовища експлуатації, або в середовищі експлуатації.

Тестування на надійність. Цей вид тестування розглядають як-основний засіб поліпшення надійності, оскільки із зростанням кількості виявлених і усунених дефектів надійність ПС зростає. Тестування на надійність виконується на наборах даних, вибраних випадковим чином з вхідного простору відповідно до операційного профілю. Всі відмови при виконанні тестів реєструються стосовно інтервалам часу між відмовами (або моментам часу настання відмов від початку тестування). Процес виникнення відмов розглядається як випадковий (стохастичний) процес (Марківський або Пуасонівський), а для кількісної оцінки досягнутої надійності застосовуються моделі зростання надійності.

Тестування продуктивності (Performance testing). Виконується для перевірки досягнення встановлених вимог до продуктивності або для оптимізації продуктивності. Інколи підрозділяється на наступні види:

- тестування (load testing) навантаження - перевірка працездатності ПС при очікуваних нормальних завантаженнях;

- тестування на стійкість (stress testing) - перевірка працездатності в нестандартних умовах (при надмірних і відсутніх завантаженнях);

- тестування об'єму (volume testing) - перевірка внутрішньо-програмних або системних обмежень, зв'язаних, наприклад, з великими масивами даних, граничними об'ємами БД і іншими характеристиками об'єму.

Тестування конфігурації (Configuration testing). Виконується для перевірки працездатності ПС в різних конфігураціях (операційних системах).

Порівняльне тестування (Hack-to-hack testing). Вид тестування, при якому один і той же набір тестів виконується на двох реалізованих версіях, а результати порівнюються. Особливо актуальний в ітераційних моделях розробки.

Тестування відновлення (Recovery testing). Проводиться для перевірки процедур відновлення системи після відмов.

Керована тестами розробка (Test-driven development). Підхід до розробки, згідно якому тести модулів створюються до початку їх кодування. Тести виконують роль деякого прототипу модуля і замінника специфікації вимог. Підхід застосовується в екстремальному програмуванні (XP).

2.3. Методи тестування

2.3.1. Класифікація і короткий огляд методів тестування

Методи тестування розрізняються підходами до проектування тестів.

Традиційно методи тестування розділяють на дві категорії - «чорний ящик» (без доступу до початкового коду) і «білий» ящик (з доступом до початкового коду).

Детальніша класифікація методів тестування, заснована на підходах до проектування тестів, представлена на рисунку 2.4. і коротко описана нижче.

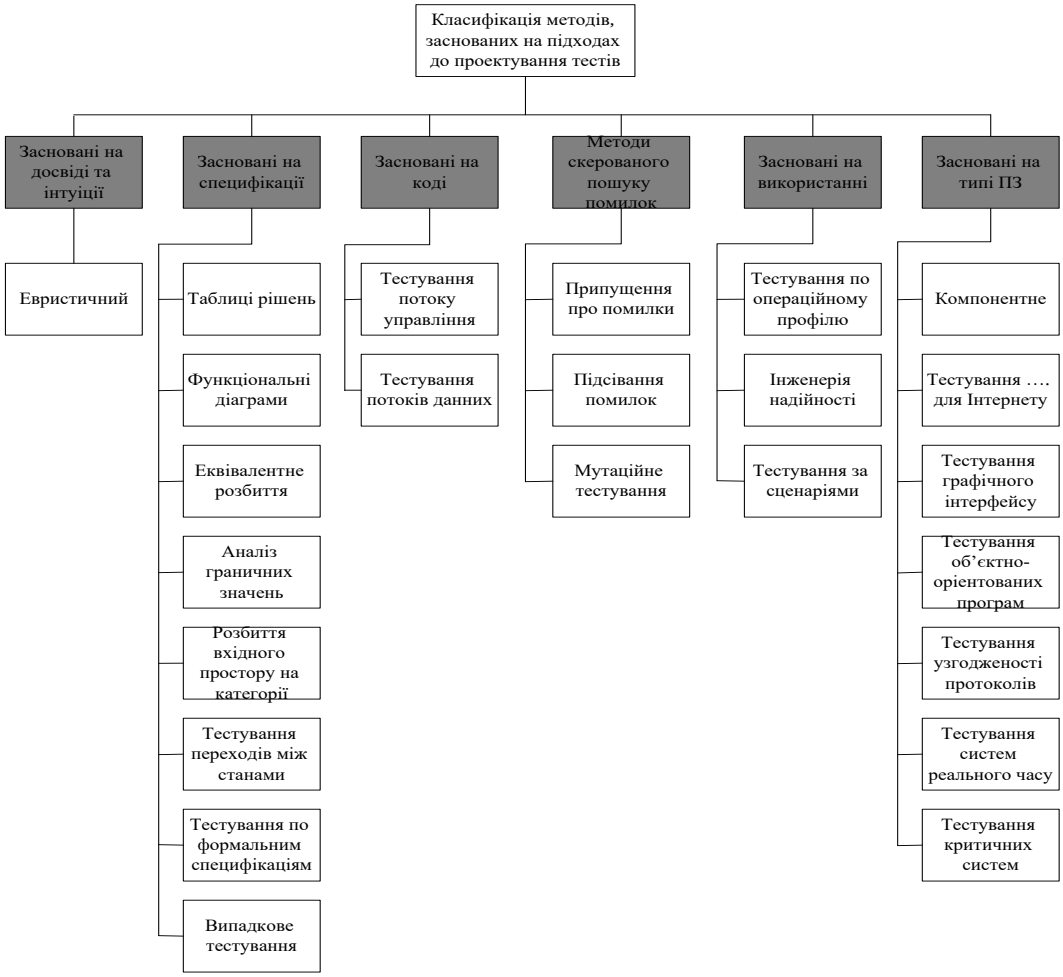

Рис. 2.4. Класифікація методів тестування

Методи тестування, засновані на досвіді і інтуїції

Спеціалізоване тестування (Ad hoc). Це найбільш поширений (хоча і вважається систематичним) підхід, при якому тести розробляються виходячи з інтуїції тестувальника і його досвіду в тестуванні подібних систем. Ефективність підходу повністю визначається майстерністю тестувальника. Підхід не вимагає детальної специфікації функцій ПЗ, але і не забезпечує оцінювання повноти тестового покриття. Розвитком підходу можна вважати «дослідницьке тестування» (exploratory testing), основні принципи якого - поєднання вивчення програмного продукту з проектуванням тестів і їх виконанням. Таке тестування зазвичай здійснюється незалежними тестувальниками стосовно завершених програмних продуктів.

Методи, засновані на специфікації (або методи функціонального тестування)

Ці методи широко відомі і у традиційній класифікації їх відносять до методів «чорного ящика». Вони застосовуються для тестування зовнішніх і внутрішніх функцій програми і передбачають наявність специфікації (формальної або не формальної), що використовується як еталон. Відрізняються між собою підходами до вибору тестових даних з множини входів (вхідного простору) функцій.

До основних методів функціонального тестування відносяться: таблиці рішень; функціональні діаграми; еквівалентне розбиття; аналіз граничних значень; розбиття вхідного простору на категорії; тестування переходів між станами; тестування по формальних специфікаціях.

Метод, що використовує таблиці рішень для проектування тестів, був запропонований Дж. Гуденафом і С. Герхартом. Кожна колонка такої таблиці представляє комбінацію умов, які можуть істотно вплинути на виконання програми. Ці умови ідентифікуються на основі аналізу специфікацій. Потім визначається множина їх можливих значень, і встановлюються обмеження відносно їх сумісності. Таким чином, скорочується кількість тестових предикатів (наприклад, обмеження може говорити про те, що, якщо умова 1 виконується, то ні умова 2, ні умова 3 не можуть виконуватися).

Другий метод, запропонований Б.Ельмендорфом і описаний Г.Майерсом, полягає в перетворенні специфікації у функціональні діаграми. По цьому методу спочатку ідентифікується кожна окрема функція, потім визначаються всі причини, що істотно впливають на її поведінку, і всі у відповідь реакції (наслідки). Наступний крок полягає в побудові графа (діаграми), що зв'язує комбінації причин з очікуваними реакціями на них. Далі для кожного слідства, вказаного на графі, визначаються тестові набори дорогою перебору всіх комбінацій причин, що породжують цей наслідок. Хоча суворе дотримання методу може забезпечити побудову ефективних тестів, він складний в практичному вживанні для

функціонально складних програм, оскільки із зростанням числа причин ускладнюється граф причинно-наслідкових зв'язків, а встановлення обмежень зв'язане з додаванням нових проміжних вузлів.

У методі еквівалентного розбиття множина значень вихідних (вхідних) даних функції, утворюючих цей вхідний простір, розбивається на набір підмножин таким чином, що в кожну підмножину потрапляють значення, еквівалентні один одному з точки зору їх використання в тестах для виявлення помилок. Говорять, що всі тести, які можуть бути побудовані на основі еквівалентних значень, представляють один «клас еквівалентності», і для тестування досить обрати лише найефективніші з них.

У методі аналізу граничних значень, який доповнює попередній метод, дані вибираються на кордонах вхідної області, оскільки багато відмов відбуваються із-за дефектів, пов'язаних з обробкою граничних значень входів. Просте і цінне розширення цього методу - тестування стійкості, коли тестові дані вибираються також і поза областю для тестування відмово-стійкості програми до недопустимих входів.

Методи еквівалентного розбиття і аналізу граничних значень вважаються базовими методами функціонального тестування і повинні застосовуватися разом при проектуванні набору тестів для кожного рівня тестування.

У розвиток цих методів був створений метод, заснований на специфікаціях функцій, і що використовує для побудови функціональних тестів розбиття вхідного простору функцій на певні категорії (category partition method або CP-метод). Суть методу полягає у ряді послідовних декомпозицій функції, починаючи з початкової функціональної специфікації, і закінчуючи окремими деталями кожної тестованої процедури, і створенні специфікацій тестів на основі виділення категорій інформації про параметри функції і умов її виконання.

У методі тестування переходів між станами програма (реалізуючи функцію) представляється у вигляді моделі, що відображає всі можливі стани її виконання, переходи між цими станами, події, переходи, і подальші дії з обробки даних, визначені цими переходами.

Опис специфікацій на формальній мові (що має точно певний синтаксис і семантику) дозволяє автоматично будувати по ним функціональні тести і в той же час забезпечує еталон для перевірки результатів. Існує цілий ряд методів генерації тестів по формальних специфікаціям.

Всі згадані методи функціонального тестування відносяться до «систематичних» методів на відміну від випадкових (стохастичних) і статистичних методів.

При випадковому тестуванні вхідні дані для тестів вибираються випадково. Як інструмент для генерації вхідних даних можуть застосовуватися датчики випадкових чисел.

Для інтерактивних програм тестувалник використовує випадкову комбінацію дій з програмою, намагаючись виявити області її нестійкого функціонування.

Методи, засновані на аналізі коду (або структурні)

У традиційній класифікації структурні методи відносять до методів «білого ящика». Згідно цим методам структура програми представляється у вигляді направленого графа, в якому ідентифікуються потоки управління або потоки даних. Відповідно, методи діляться на дві основні категорії: тестування потоку управління (шляхів) і тестування потоку даних.

У методах тестування потоку управління дані з вхідного простору вибираються так, щоб забезпечити максимальне покриття коду. Основні методи приведені нижче:

Тестування рядків. Вибираються дані, що забезпечують виконання всіх рядків (операторів) програми. Цей метод дасть найслабший критерій покриття - «покриття рядків», прийнятний для програм, що не містять логічних умов і циклів.

Тестування гілок. Вибираються дані, що забезпечують виконання шляхів, що виділяються в програмі за допомогою логічних умов, що набувають значень Правда або Неправда .

Тестування логічних умов. Якщо гілки в програмі утворюються в результаті виконання складних логічних умов, дані для їх тестування повинні вибиратися так, щоб перевірити всі значення логічних умов.

Цей метод дає найповніший критерій покриття коду програми, котрий називається критерієм «логічні умови». Знову таки, для задоволення критерію покриття рядків було б досить одного тесту, а для покриття гілок - два.

Тестування циклів. Тести розробляються для перевірки кожного циклу при граничних значеннях змінних циклу і усередині їх границь (для тестування стійкості цикли перевіряються при значеннях поза границею). Цей метод дає критерій покриття - «всі цикли».

У методі тестування потоку даних тестові дані вибираються таким чином, аби прослідкувати шлях кожної змінної в програмі від визначених значень до останнього використання (для всіх змінних). Цей метод вимагає великої кількості тестів, тому на практиці трасуються лише найбільш критичні значення змінних. Особливий інтерес, наприклад, представляють значення змінних, що беруть участь в операціях ділення, умови і цикли, виконання яких залежить від значень змінних. Метод тестування потоку даних може також застосовуватися для пошуку помилок часу виконання, що важко виявляються, зв'язаних з використанням пам'яті.

На закінчення короткого огляду структурних методів потрібно відмітити, що вони зазвичай застосовуються на рівні модульного тестування програми її розробником і

чергуються з налагодженням, хоча можуть використовуватися і в процесах V&V для перевірки критичних програм.

Методи функціонального і структурного тестування повинні розглядатися не як альтернативні, а як взаємодоповнюючі, оскільки використовують різні джерела інформації для побудови тестів і призначені для виявлення різних типів помилок.

Методи направленого пошуку помилок

Ці методи використовуються для проектування тестів, спеціально створених для виявлення помилок певних типів. До основних методів цієї категорії відносяться: припущення про помилки (error guessing); підсівання помилок (error seeding); мутаційне тестування (mutation testing).

Згідно методу висунення припущень про помилки на підставі «історичної» інформації про помилки, виявлені в подібних програмах, досвіду тестувальника, а також каталогів відомих помилок, складається список можливих помилок і помилкових ситуацій. Одним з «класичних» прикладів вживання методу є перевірка ділення на 0. У традиційній класифікації метод відноситься до категорії методів «чорного ящика». Розглянутий підхід до тестування і налагодження програм, написаних на мові Java, заснований на аналізі «шаблонів» типових помилок. Основна його ідея – вчитися на чужих помилках, аби не допускати власних.

Методи підсівання помилок і мутаційного тестування призначені для перевірки ретельності вже виконаного тестування. У традиційній класифікації вони відносяться до категорії методів «білого ящика».

У методі підсівання помилок в код, протестований на певному наборі тестів, спеціально вноситься невелика кількість помилок, а потім програма повторно тестується. Якщо при тестуванні виявляються не всі внесені помилки, набір тестів вважається не достатнім. Відношення кількості виявлених внесених помилок до загальної кількості внесених помилок припускається приблизно рівним відношенню кількості знайдених реальних помилок до загального числа помилок, що містяться в програмі. Якщо виявлені всі внесені помилки, то вважається, що, або набір тестів достатній, або, що ці помилки було дуже легко знайти.

При мутаційному тестуванні створюється багато копій основної програми, до кожної з яких вноситься невелика зміна, зване «мутацією», для імітації помилки в операторі або, що не відбивається на синтаксичній коректності програми. Кожна копія тестується на одному і тому ж наборі тестів. Якщо в процесі тестування всі внесені зміни виявлені, програма вважається протестованою адекватно і коректною (можуть бути також виявлені раніше не виявлені помилки). Для того, щоб метод був ефективним у виявленні помилок, потрібна велика кількість мутантів, що можливо лише при автоматичній їх генерації.

Методи, засновані на аналізі очікуваного використання

До цієї категорії в віднесено два основні методи: статистичне тестування; тестування за операційним профілем.

При статистичному тестуванні вхідні дані для проектування тестів вибираються з вхідного простору відповідно до частоти їх появи в майбутніх сценаріях використання програми. При виконанні тестів фіксуються моменти відмов і обчислюються інтервали між відмовами MTBF зазвичай в одиницях процесорного часу (CPU) або часу виконання. Отримана інформація використовується для оцінки надійності передбачення моменту завершення тестування. Метод застосовується на рівні системного тестування. Для визначення частоти використання функцій програми, а також моментів відмов, в код вставляються команди-лічильники.

Тестування за операційним профілем застосовується в рамках методології інженерії надійності (SRE) і називається методом SRET. Методологія SRE охоплює повний цикл розробки програмних систем з підвищеними вимогам до надійності, а тестування SRET (що є по суті статистичним) основане на пріоритеті входів не лише по частоті, але і з врахуванням критичності функцій, режимів і наслідків відмов систем при експлуатації.

До категорії методів, заснованих на аналізі очікуваного використання, можна віднести також метод тестування сценарієм можливого використання, суть якого полягає в ідентифікації можливих сценаріїв роботи користувача і розробці по ним сценаріїв тестування (test-cases). Цей метод застосовується у всіх сучасних інструментах автоматизації функціонального тестування програм, що мають інтерфейс користувача (інструменти Capture-Replay). Він також використовується в популярній методології RUP, де сценарії тестування формуються на етапі аналізу і проектування по набору «бізнес-сценаріїв» (use-cases).

Узагальненням даного методу можна вважати тестування на базі моделей (model-based testing, model-driven testing), вживане в рамках відповідного підходу до розробки (model-driven development).

Такі відомі підходи до тестування як тестування, засноване на ризику (risk-driven testing, risk-based testing), представляють не методи розробки тестів, а стратегію направленого тестування і мінімізації набору тестів. Ці стратегії подібні до тестування за операційним профілем, але проводяться методами систематичного тестування і враховують не лише частоту використання, але і величину ризику відмови ПС.

Методи, що зважають на специфіку програмної системи

Розглянуті вище методи універсальні і застосовні до будь-яких типів ПС. Проте вони не враховують характерних особливостей побудови систем різного типу, що вимагають застосування специфічних підходів до тестування.

Керуючись рекомендаціями SWEBOK (2004), можна охарактеризувати наступні методи: тестування об'єктно-орієнтованих програм; компонентне тестування; тестування Web-застосувань; тестування графічного інтерфейсу користувача; тестування протоколів на відповідність; тестування систем реального часу; тестування критичних систем.

Особливості типів ПС обумовлюють не стільки вживання спеціальних методів проектування тестів, скільки вибір методів і видів тестування, найбільш відповідних для певних об'єктів тестування.

2.3.2. Дослідницьке тестування

Методи дослідницького тестування призначені для вивчення програми і виявлення в ній помилок одночасно. Ці методи застосовуються в стратегіях тестування, заснованих на ризику.

Один з методів дослідницького тестування і процедура його застосування для тестування функціональності і стійкості програмних продуктів в середовищі Windows.

Процедура включає наступні кроки:

Крок 1. Дослідження. Вивчення призначення і функцій програмного продукту, типів оброблюваних даних і областей його можливої нестійкості (чинників ризику відмови).

Успіх виконання кроку залежить від наявності інформації про програмний продукт і його потенційних користувачів, а також час для його вивчення. Для здобуття інформації про продукт може використовуватися документація (у тому числі Help), відомості про аналогічні продукти, вивчення продукту шляхом його короткочасного використання. Завдання кроку:

1. Формування списку функцій (ієрархії функцій).
2. Розбиття функцій на основні і другорядні. У таблиці 2.1 наведений приклад критерію розбиття по важливості.
3. Виявлення областей можливої нестійкості продукту (схильні до відмов функцій і даних).

Таблиця 2.1. Приклад розбиття функцій на основні та другорядні

Визначення	Критерії класифікації
Основна функція – будь-яка зовнішня функція (видима користувачу), відмови в якій означають невідповідність продукту своєму призначенню.	1. Функції, котрі є визначальними для використання продукту за призначенням. 2. Функції, які часто застосовують звичайні користувачі в сеансі роботи. 3. Функції, які використовуються не часто, але їх відмови призводять до значних негативних наслідків.
Другорядна функція – функція, без якої продукт може використовуватися.	Функції, які використовуються не часто, відмови при виконанні яких не можуть призвести до серйозних наслідків.

Приклади областей можливої нестійкості функцій: функції обробки зовнішніх подій; функції, що інтенсивно використовують оперативну пам'ять; дуже складні функції; функції,

що використовують засоби Windows і що змінюють її параметри (налаштування параметрів); функції, що маніпулюють конфігурацією Windows; функції аналізу вхідних даних і обробки помилок; функції, що підміняють базові функції Windows (наприклад, відновлення видалених файлів); функції або групи функцій, що використовують багато паралельних процесів; функції, що працюють з багатьма файлами одночасно; функції, що працюють з файлами, розташованими в мережі.

Приклади областей можливої нестійкості при обробці даних:

- документи: довгі, багато одночасно відкритих;
- записи: довгі, багато записів, порожні, складні;
- списки: довгі, порожні або багатоколоночні;
- поля: введення великої кількості символів, дуже великі значення числових полів;
- об'єкти: багато, великі, складні і ін.

Крок 2. Проектування тестів. Визначення стратегій виконання тестів і критеріїв оцінювання результатів (як, наприклад, в таблиці 2.2).

Таблиця 2.2. Приклад критеріїв проходження тестів

Визначення мети	**Критерій проходу**	**Критерій відмови**
Функціональність - спроможність продукту виконувати необхідні функції.	Функція виконується в відповідності з її призначенням.	Хоч би одна функція не виконується у відповідності з її призначенням
	Будь-яке неправильне виконання функції не призводить до серйозних наслідків при коректному використанні.	Неправильне виконання призводить до серйозних наслідків навіть при коректному використанні.
Стійкість – спроможність продукту функціонувати без відмов при підвищених навантаженнях (по часу, пам'яті).	Робота продукту при підвищених навантаженнях не призводить до руйнування Windows не призводить до втрати даних, зависанню, відмові. При тестуванні не було виявлено відмов або будь-яких дефектів при виконанні основних функцій.	Робота продукту руйнує Windows. Робота продукту може привести до втрати даних, зависанню, відмові. При тестуванні хоча б одної з основних функцій постерігалися відмови.

Крок 3. Виконання тестів. Виконання тестів, спостереження і фіксація результатів і використання цієї інформації для формування думки про роботу продукту.

Завдання кроку: тестування всіх основних функцій; тестування ідентифікованих областей потенційної нестійкості; вибіркове тестування другорядних функцій; реєстрація відмов; фіксація будь-яких виявлених проблем (для подальшого вивчення).

Крок 4. Побудова евристик. Евристики представляють неформальні правила (засновані на досвіді тестувальника і здоровому глузді) ухвалення рішення про те, чи вважати роботу продукту прийнятною чи ні, і як далі продовжувати тестування.

Крок 5. Визначення критерію покриття. У даній процедурі використовується наступний критерій покриття: протестовані всі основні функцій; протестовані вибрані другорядні функції; протестовані вибрані області можливої нестійкості (слід вибрати від п'яти до десяти функцій і протестувати їх на тестових даних, які можуть привести до нестійкої роботи продукту).

Розподіл часу тестування, що рекомендується: 80% часу - на основні функції. 10% - на другорядних і 10% – на області потенційної нестійкості. Процедура тестування по даному методу виконується циклічно.

2.3.3. Еквівалентне розбиття

Суть методу еквівалентного розбиття полягає в тому, що вся множина тестів, які теоретично можна побудувати для тестування програми, яка реалізує функцію, розбивається на підмножини тестів, утворюючих «класи еквівалентності», з яких для виконання вибираються лише найбільш представлені (з найбільшою вірогідністю виявлення помилки).

Віднесення групи тестів до одного класу еквівалентності може засновуватися на наступних практичних критеріях: тести включають значення одних і тих же вхідних даних; при запуску тестів виконуються одні і ті же операції; від виконання тестів очікуються однакові результати; або жоден з тестів не викликає виконання блоку обробки помилок, або всі тести викликають виконання цього блоку (у припущенні, що програма взагалі містить блоки обробки помилок).

Класи еквівалентності підрозділяються на допустимі класи еквівалентності, що представляють правильні вхідні дані і вхідні події (події-входи), і недопустимі класи еквівалентності, представляючи всі останні дані або події. Для представлення класів еквівалентності можна скористатися табличною формою (її приклад в таблиці 2.3).

При виділенні класів еквівалентності можна керуватися рядом загальних правил:

- якщо подія-вхід асоціюється з областю значень (наприклад, як в таблиці 2.3), то визначається один допустимий клас еквівалентності і ряд недопустимих (у таблиці їх п'ять);
- якщо подія-вхід асоціюється із значенням, що представляє кількість чого-небудь (наприклад, від 1 до 20), то визначається один допустимий клас еквівалентності (1<= X <= 20) і ряд недопустимих (X < 1 і X > 20);
- якщо для параметра програми допускається лише визначений перелік значень (наприклад, перелік конкретних найменувань чого-небудь, збережених в базі даних), то визначається один допустимий клас еквівалентності для значень з цього переліку і один недопустимий (наприклад, для значень, відсутніх в переліку). В цьому випадку граничні значення не визначаються;

- будь-який елемент списку параметрів програми або об'єктів (списки, меню, кнопки панелі інструментів) представляє окремий клас еквівалентності.

Таблиця 2.3. Приклад опису класів еквівалентності

Вхідні події	**Допустимі класи еквівалентності**	**Недопустимі класи еквівалентності**
Введення числа	Числа від 0 до 9000	Числа менші 0 та більше 9000 Пробіл Нечислові символи Обчислюваний вираз, результат якого знаходиться за межами допустимого інтервалу
Введення першої літери прізвища	Перший символ має бути великою літерою	Перший символ маленька літера Перший символ – не літера

Тести розробляються спочатку для допустимих класів еквівалентності, а потім для недопустимих:

- для допустимих: по одному тесту усередині класу і по одному на границі класу (наприклад, якщо параметр програми містить список, потрібно із списку вибрати перший, останній і будь-який проміжний елемент, а для числових значений в діапазоні від 1 до 100, вибрати 1, 100 і будь-яке число усередині діапазону);

- для недопустимих: по одному тесту для кожного класу (наприклад, для числових значень в наведеному вище прикладі - вибрати 0 і 101, пропуск і нечисловий символ).

2.3.4. Аналіз граничних значень

По цьому методу тести розробляються для направленого тестування границь класів еквівалентності. Якщо область вхідних даних класу представляє безперервний діапазон значень, то вибираються допустимі значення, розташовані на нижньому і верхньому кордонах діапазону. Якщо діапазон дискретний (наприклад, елементи списку значень) – вибирається перший і останній допустимі елементи.

При цьому аналізуються також всі вихідні класи еквівалентності - очікувані результати тестів. Якщо очікувані результати представляють значення з безперервного діапазону значень, то вхідні дані для їх здобуття вибираються так, щоб набути значень, що знаходяться на верхній і нижній границях допустимого діапазону. В тому випадку, якщо результати є наборами значень результатів, вхідні дані вибираються для здобуття першого і останнього елементів (наприклад, перевірити вивід на друк першого і останнього рядка звіту).

Відмінність даного методу від методу еквівалентного розбиття полягає в наступному:

- при аналізі граничних значень вибираються лише такі елементи в класі еквівалентності, які дозволяють перевірити тестом кожну границю цього класу;

- при розробці тестів розглядаються не лише вхідні значення, але і очікувані результати.

2.3.5. Розбиття вхідного простору на категорії

Процедура вживання цього методу включає наступну послідовність кроків.

Крок 1. Декомпозиція функції на функціональні елементи, які можуть тестуватися незалежно. Для великих і складних функцій декомпозиція виконується ієрархічно. З кожним функціональним елементом зв'язується один або декілька модулів, що його реалізують. Побічним позитивним ефектом виконання цього кроку є аналіз якості декомпозиції проекту по функціональній ознаці.

Крок 2. Ідентифікація параметрів і умов середовища, що впливають на поведінку функції. Параметри є входами до функціонального елементу, а умови середовища визначають спосіб реалізації функціонального елементу і характеристики стану системи під час його виконання. Наприклад, для функцій інтерфейсу користувача параметрами можуть бути поля введення, поля виводу, список обраних елементів і ін., а умовами середовища – файли, поля таблиці бази даних і ін. Реальні тести для функціонального елементу являються композицією певних значень параметрів і умов середовища, вибраних з метою максимізації шансів пошуку дефектів в реалізації елементу.

Крок 3. Виділення категорій інформації, що характеризує кожен параметр і умови середовища. Категорія є основною властивістю (або характеристикою) параметра або умови середовища. У наведеному вище прикладі параметрів і умов середовища для функцій інтерфейсу користувача можна виділити наступні категорії:

- для параметра «поле введення» – категорії тип, розмірність, стан, наповнення, обов'язковість і ін.;
- для умови середовища «вікно» – категорії координати, колір вікна, колір символів, рамка і ін.

Вибір категорій виконується шляхом пошуку в специфікації функції ключових фраз, які описують, як поводиться функціональний елемент при певних поєднаннях параметрів і умов середовища.

Крок 4. Розбиття кожної категорії на чіткі альтернативи, які включають різні види значень, можливі для категорії. Альтернативи являються класами розбиття, з якого вибиратимуться представницькі елементи для побудови тестів. У таблиці 2.4 наведені приклади категорій і можливих альтернатив стосовно згаданих параметрів функцій інтерфейсу користувача.

Крок 5. Формування формальної специфікації тесту для кожного функціонального елементу. Специфікація тесту складається із списку категорій і списків альтернатив в кожній

категорії. Інформація із специфікації тесту використовується далі для отримання множини фреймів тесту, що є основою для створення реальних тестових наборів. Фрейм тесту складається з множини виділених альтернатив, причому кожна категорія представлена не більше ніж однією альтернативою (або нулем). Побудована специфікація тесту практично не обмежена, оскільки в ній не враховані можливі стосунки між альтернативами. Повне число фреймів, що згенерували по такій специфікації, дорівнює добутку числа альтернатив в кожній категорії. З метою скорочення числа можливих тестових фреймів встановлюються обмеження для взаємозв'язаних альтернатив.

Таблиця 2.4. Приклад розбиття на категорії та альтернативи

Параметр	**Категорії**	**Альтернативи**
Поле вибору	Зміст	Текст Піктограма Значок з текстом
	Стан	Поле не вказано/ вибрано Поле не вказано/ не вибрано Поле вказано/ вибрано Поле вказано/ не вибрано
	Тип поля	Однозначний вибір Багатозначний вибір Розширений вибір
Меню	Спосіб вибору	Вибір курсором Виділена літера/(комбінація літер) Функціональна клавіша Вибір за допомогою клавіатури

Типове обмеження вказує на те, що альтернатива з однієї категорії не може з'явитися разом в тестовому фреймі з певними альтернативами з інших категорій.

Крок 6. Генерація тестових фреймів. Оскільки процес формування тестових фреймів пов'язаний з перебором великого числа комбінацій альтернатив з урахуванням обмежень, він має бути по можливості, автоматизований. Тестовий фрейм, що генерується, міститиме по одній альтернативі для кожної категорії.

Крок 7. Перетворення тестових фреймів в тестові набори і розробка тестових сценаріїв. Реальні тестові набори для функціонального елементу є композицією певних значень параметрів і умов середовища, представлених альтернативами відповідних категорій і підібраних з метою максимізації шансів виявлення дефектів в реалізації функціонального елемента.

Тестовий сценарій формується з послідовності зв'язаних тестів для одного або більше функціональних елементів, що вимагають однакових умов середовища.

Цей метод володіє рядом очевидних переваг:

• дозволяє охопити відразу два основні аспекти тестування – перевірку повноти реалізації функцій і виявлення різних класів дефектів в найбільш уразливих частинах програми;

• функціональний аналіз є невід'ємною частиною методу і виконується паралельно із специфікацією функціональних вимог (або відразу після), забезпечуючи своєчасне усунення дефектів специфікацій;

• хоча специфікація тесту повинна охоплювати всі можливі категорії інформації і варіанти поєднання альтернатив, застосовуючи механізм обмежень і керуючись практичними міркуваннями, можна управляти об'ємом тестування;

• процес перебору альтернатив і «відсіювання» неможливих або небажаних їх поєднань може бути автоматизований, що позбавить тестувальника від рутинної роботи.

2.3.6. Тестування переходів між станами

По цьому методі тести проектуються для перевірки переходів між станами програми (наприклад, зміни зображення на екрані, зміни складу і активності елементів меню і ін.). Оскільки кількість можливих станів і переходів між ними може бути значно більше, ніж можна протестувати, при проектуванні тестів слід керуватися практичними міркуваннями:

• набір тестів повинен охоплювати найбільш вірогідні послідовності дій користувачів;

• якщо є залежні стани і події, що синхронізуються, для кожного з них повинні розроблятися окремі тести;

• окремі тести слід проектувати також для перевірки недопустимих переходів між станами (для виявлення можливих небажаних побічних ефектів);

• після виконання мінімального набору тестів слід провести набір випадкових тестів (вибираючи довільні режими і сценарії виконання).

Як інструменти при проектуванні тестів використовують: діаграми переходів в стани (State-transition diagram); таблиці переходів в стани (State-transition tables).

Діаграма переходу в стани відображає множину станів в контексті, події, які викликають переходи між станами, і можливі дії. Таблиці переходів в стани в табличному вигляді (таблиця 2.5).

Таблиця 2.5. Структура таблиці переходів станів

Стан	**Подія**	**Дія**	**Наступний стан**

Цей метод спочатку призначався для тестування програм реального часу, але згодом став застосовуватися при проектуванні тестів для інтерактивних програм: тестування станів

меню (активно, не активно, вибрано), навігації між вікнами, синхронізації елементів інтерфейсу в діалогових вікнах.

2.4. Аналіз результатів тестування

2.4.1. Система відстеження проблем

На ефективність тестування і усунення дефектів робить великий вплив міра чіткості процесу підготовки і аналізу звітів про проблеми, а також наявність хорошого інструменту для його підтримки. За відсутності промислового інструменту, група тестування може створити свою систему відстеження проблем, наприклад, за допомогою MS Access.

Цілі системи відстеження проблем:

- відстеження стану тестування і усунення дефектів;

- організація взаємодії між співробітниками і вирішення спірних питань відносно класифікації і пріоритетів усунення дефектів;

- визначення причин дефектів і виявлення «вузьких місць» в процесах розробки і тестування.

Із звітами про проблеми в процесі тестування працюють тестувальники, розробники, керівник проекту і група якості. Тому для ефективного вирішення проблеми важливо визначити повноваження користувачів системи відстеження проблем (як виконавців процесу «Вирішення проблем»).

Процес вирішення проблем, виявлених при тестуванні, може бути представлений наступною послідовністю кроків:

Крок 1. Відкриття проблеми. При виявленні проблеми тестувальник складає звіт про проблему («відкриває проблему») і передає його програмістові для аналізу. При автоматизованому веденні звітів тестувальник поміщає звіт в базу даних.

Крок 2. Вивчення. Програміст аналізує звіт і приймає рішення (згоден чи ні). Якщо проблема викликана випадковою відмовою, зв'язаною, наприклад, із збоєм устаткування, середовищем тестування, порушеннями технології тестування або нерозумінням тестувальника – вона відхиляється. Якщо встановлено, що проблема пов'язана з дефектом в ПЗ, програміст встановлює пріоритет усунення дефекту. У спірних випадках рішення про пріоритет усунення приймає керівник проекту.

Крок 3. Дія. Програміст виконує усунення дефекту і повторне автономне тестування і повертає звіт з відміткою «Виправлений» тестувальнику. При автоматизованому веденні звітів тестувальник знаходить звіти з цією відміткою в базі даних.

Крок 4. Закриття. Тестувальник виконує перевірку виправлень і закриває проблему. Закрити звіт про проблему може лише тестувальник. При автоматизованому веденні звітів про проблеми вони зберігаються в базі даних для подальшого аналізу, формування звітів, а також

для накопичення історичних даних. Звіти про відхилені дефекти можуть віддалятися з бази даних (але лише тестувальником!).

Кроки 1-4 стосуються відстеження кожного окремого дефекту, але для аналізу результатів і управління процесом тестування використовуються узагальнені і класифіковані дані про дефекти. Цей аналіз і узагальнення виконуються менеджером групи тестування і менеджером проекту. Узагальнені дані використовуються також групою якості для аналізу поточного стану проекту.

2.4.2. Класифікація дефектів, виявлених при тестуванні

Існують різні підходи до класифікації дефектів, які вносяться до ПС і виявляються на різних стадіях її розробки. Найповніша класифікація представлена в стандарті IEEE Std. 1044:1993. У нім замість терміну дефект використовується термін аномалія для позначення будь-яких відхилень в ПС, її специфікаціях, документації, а також планах або процедурах, пов'язаних з розробкою або використанням ПС. У цьому розділі розглядаються питання класифікації дефектів, які виявляються при тестуванні.

Найбільш важливими аспектами класифікації дефекту є: симптом; серйозність; пріоритет усунення; стадія розробки і джерело; тип.

Симптом дефекту стосується його видимого прояву, що спостерігає тестувальник при виконанні тестів. Опис симптому необхідний для аналізу дефекту розробником і визначення дійсної причини дефекту. У IEEE Std. 1044, що рекомендується, класифікація симптомів така:

- Аварійне завершення програми;
- Неочікувана поведінка програми;
- "Зависання" програми;
- Проблема введення:
  - Коректні дані не вводяться;
  - Неправильні дані вводяться;
  - Опис даних відсутній або неправильно;
  - Параметри не повні або відсутні;
- Проблема виводу:
  - Неправильний формат;
  - Неправильні результати/данні;
  - Вивід не повний або відсутній;
  - Граматика/синтаксис;
- Незадовільна продуктивність;
- Відчуття загальної відмови продукту;
- Повідомлення про помилку системи;
- Інше.

Для класифікації дефектів за серйозністю стандарт IEEE Std.1044: 1993 рекомендує таку порядкову шкалу:

- Критичний
- Серйозний
- Значний
- Незначний
- Не дефект

У кожному конкретному випадку «серйозність» дефекту залежить від типу системи і повинна визначатися в контексті наслідків дефекту для системи (або для користувача). Приклад віднесення дефекту до категорії серйозності представлений в таблиці 2.6 (у ній кожному рівню серйозності привласнений код).

Таблиця 2.6. Опис серйозності дефекту

Код	**Серйозність**	**Опис**
1	Критичний	Дефект призводить до відмови всієї системи, підсистеми, компонента. Подальше тестування і/або використання системи неможливе.
2	Серйозний	Дефект призводить до відмови компонента ПЗ, втраті даних.
3	Значний	Дефект призводить до отримання некоректних, неповних, неправдивих результатів, або дефект стосується зручності використання системи.
4	Незначний	Дефект не призводить до відмови, не погіршує зручність користування системою, його можна обійти.
5	Не дефект	Помилки в тесті, збій програмного чи апаратного середовища.

З кожним дефектом пов’язують пріоритет усунення. Для класифікації дефектів також може використовуватися порядкова шкала (таблиця 2.7)

Таблиця 2.7. Опис пріоритетів усунення дефектів

Код	**Пріоритет**	**Опис**
1	В першу чергу	Подальша розробка і/або тестування не може виконуватися до усунення дефекту.
2	Звернути особливу увагу	Дефект має бути усунутий якомога швидше, оскільки його наявність негативно впливає на розробку і/або тестування.
3	В порядку черги	Дефект має усуватися в порядку планових дій розробки. Його усунення може бути відкладене до випуску нової версії.
4	Відкласти	Усунення дефекту може бути відкладене на невизначений термін. Він може бути усунутий в наступній версії.
5	Відмінити	Не дефект. Можливо випадковий збій чи помилка тестувальника.

Класифікація дефектів за стадіями розробки співвідносить дефекти із стадіями розробки (і процесами), на яких вони були внесені.

Класифікація дефектів за джерелами співвідносить дефекти з робочими продуктами стадій розробки, використання яких привело до появи дефектів в коді ПЗ, наприклад:

- Специфікація (вимог, функцій, проекту, інтерфейсу, даних);
- Код (дефекти кодування);
- Документація на систему;
- Плани і процедури тестування.

Нижче перераховані основні типи дефектів, IEEE Std. 1044: 1993.

- Логічні
- Обчислень
- Інтерфейсу
- Обробки даних
- Введення даних
- Обробки помилок і ін.

Зібрані і класифіковані дані про дефекти складають основу для обчислення метрик тестування.

2.4.3. Вимірювання результатів тестування

Кількісне вимірювання результатів тестування ґрунтується на застосуванні метрик оцінювання поточного стану об'єктів тестування (метрики продукту) і процесу тестування (оцінка виконаних тестів). До основних метрик оцінювання продукту можна віднести метрики, представлені нижче. Метрики підрахунку дефектів, вказані в таблиці 2.8.

Таблиця 2.8. Метрики підрахунку дефектів

Метрика	**Позначення**	**Опис**
Кількість дефектів	Дфакт	Сума усіх дефектів, виявлених при тестуванні за вибраний проміжок часу. Обчислюється по відкритих і закритих звітах про проблеми.
Щільність дефектів	Щл_Дфакт	Обчислюється у вигляді відношення кількості виявлених дефектів до розміру ПЗ (KSLOC або FP). Щл_Дфакт = Дфакт/Розмір

Метрики надійності, представлені в таблиці 2.9. Ці метрики обчислюються за даними про відмови і вимагають окрім підрахунку відмов (дефектів) вимірювання інтервалів часу між відмовами.

Таблиця 2.9. Метрики надійності

Метрика	**Позначення**	**Опис**
Інтенсивність відмов (Failure Rate)	FR	Кількість відмов в одиницю часу (наприклад в годину) FR = Дфакт/T де Дфакт – кількість виявлених відмов (дефектів), T - період спостереження, виражений в вибраних одиницях часу (наприклад, в годинах).
Середній між (Mean Beetween Failure)	MTBF	Відношення суми відмов MTBF = Tc/ Дфакт де Tc - сума інтервалів часу між виявленими відмовами.

Метрики процесу тестування призначені для визначення стану виконання тестування (метрики покриття) і динаміки виявлення дефектів.

Метрики покриття служать індикаторами повноти виконаного тестування відносно вибраних критеріїв покриття і обчислюються у вигляді відношення кількості виконаних тестів до необхідної кількості. Ці метрики підрозділяються на структурні і функціональні.

Метрики структурного покриття обчислюються для вибраного структурного критерію як відношення кількості виконаних рядків коду (гілок або логічних умов) до загальної їх кількості в модулі.

Метрики функціонального покриття обчислюються як відношення кількості виконаних функціональних тестів до кількості тестів, потрібних для задоволення критерію, що відповідає вибраному методу функціонального тестування.

2.4.4. Критерії завершення тестування

Як відомо, найбільш поширений критерій завершення тестування – це вичерпаний час, виділений на його проведення. Цей критерій ніяк не враховує виконаного об'єму тестування і ризики відмови ПС із-за дефектів, що залишилися. Більш суворі критерії ґрунтуються на кількісних вимірах.

У підході, заснованому на метриках покриття, критерії формуються шляхом обчислення метрик функціонального і структурного покриття і відображають об'єм виконаного тестування. Структурні критерії застосовуються при автономному тестуванні і засновані на методах структурного тестування, а функціональні застосовуються на всіх рівнях і засновані на методах функціонального тестування.

Згідно підходу, заснованому на профілі дефектів, тестування припиняється, якщо немає нових і відкритих дефектів серйозності 1, 2, 3. Цей критерій застосовується при функціональному і системному тестуванні.

Згідно підходу, заснованому на оцінках інтенсивності відмов, тестування продовжується до тих пір, поки не будуть досягнуті встановлені у вимогах значення метрик надійності (інтенсивність відмов і середній час роботи без відмови). Критерій застосовується на рівні системного тестування і передбачає статистичне тестування (за операційним профілем).

Оскільки жоден з критеріїв не гарантує повноти тестування, при ухваленні рішення про завершення тестування необхідно використовувати комплексні критерії. Наприклад, в роботі сформульований комплексний критерій завершення тестування для інформаційних систем:

- всі заплановані функціональні тести пройшли (Тплан - 100%);
- структурне тестування було виконане набором тестів, який забезпечив 100% покриття рядків, 80% покриття логічних умов і 100% покриття викликів процедур;

• немає відкритих дефектів серйозності 1, 2 і 3 і щільність дефектів нижча, ніж 0.5 дефектів на KSLOC;

• інтенсивність виявлення відмов не вище 40 нових відмов на 1000 годин тестування;

• тривалість безперервного функціонування ПС без відмови досягає 100 годин.

В умовах обмежених ресурсів на тестування критерій завершення може бути сформульований виходячи з оцінок ризику відмови ПС. Він враховує, які ідентифіковані ризики усунені шляхом тестування, і яка серйозність дефектів, що залишилися. Згідно даному критерію, тестування може бути завершене, якщо всі відомі дефекти серйозності 1, 2 і 3 закриті, нові – не виявлені, а ризик відмови із-за дефектів, що залишилися, настільки малий, що подальше тестування економічно не вигідно.

2.5. Дослідження методів тестування програмних модулів обробки польотної інформації

2.5.1. Програмні комплекси контролю польотів

У дипломній роботі розглядається технологія верифікації програмних систем на прикладі верифікації та тестування автоматизованих систем контролю польоту. Програмне забезпечення контролю польотів складається з трьох основних програмних комплексів:

- Комплекс програм відтворення польотної інформації;

- Комплекс програм допускового контролю польоту;

- Комплекс програм контролю якості виконання польоту.

Комплекс програм відтворення ПІ вирішує задачу обробки копії польоту, яка полягає в графічному і табличному поданні зміни аналогових параметрів і разових команд у часі з можливістю аналізу як усієї копії, так і її фрагментів. Відповідний комплекс програм включається в будь-яку АСКП і складається з ряду програм і підпрограм.

Комплекс програм допускового контролю забезпечує реалізацію всіх алгоритмів контролю, визначених в експлуатаційній документації на ЛА. Результати рішення задач контролю використовуються для прийняття висновку про знаходження об'єкта контролю "ЛА-екіпаж-середовище" в тому чи іншому стані. Програми комплексу призначені для виявлення небезпечних відхилень у роботі систем ЛА і помилок пілотування, які можуть впливати на безпеку польоту

Комплекс програм контролю якості польоту є програмною системою, яка служить для обчислення показників якості як кожного елемента польоту, так і всього польоту в цілому. Для вирішення завдань контролю якості виконання польоту програмне забезпечення комплексу повинне володіти такими функціями:

- Формування інформаційного "портрета" етапів польоту;

- Визначення значень показників якості пілотування.

2.5.2. Тестування модулів обробки ПІ

У даному розділі наводиться короткий огляд методів тестування та даються рекомендації по використанню кожного з розглянутих методів при тестуванні програмних модулів, що входять до складу різних комплексів контролю польотів.

Як правило, методи тестування програм, як окремих модулів, відносять або до стратегії "чорного ящика", або до стратегії "білого ящика".

При використанні стратегії "чорного ящика" програма розглядається як "чорний ящик". Є ще два синоніма, що позначають цю стратегію: тестування з управлінням за даними та тестування з управлінням по входу–виходу.

При тестуванні програми за методом "чорного ящик", на її вхід подаються різні тестові набори, а на виході проводиться перевірка: чи не відбулося відхилень по вихідним даним програми від описаних в її специфікаціях.

Навпаки, при тестуванні програм по стратегії "білого ящика" програма розглядається як "білий ящик", а стратегія має другу назву – "тестування, кероване логікою програми". Для проведення тестування з стратегії "білого ящика", необхідно мати тексти програми на мові високого рівня, а тестові дані виходять шляхом аналізу логічної організації програми.

До стратегії "чорного ящика" відносяться наступні методи: еквівалентне розбиття; аналіз граничних значень; метод функціональних діаграм; метод припущення про помилку; динамічне тестування; стохастичне тестування.

До стратегії "білого ящика" належать такі методи: покриття операторів; покриття рішень/умов; комбінаторне покриття умов; статичне тестування; формальна верифікація; доказ правильності програм.

При використанні методу еквівалентного роздроблення проводиться розбиття вхідної області даних програми на деяке кінцеве число класів еквівалентності таким чином, що якщо один тест класу виявляє помилку, то й інші члени класу повинні виявляти цю ж помилку (класи еквівалентності не перетинаються).

Метод аналізу граничних умов є логічним розвитком методу еквівалентних розбиттів і характеризується такими положеннями:

1) елементи класу еквівалентності вибираються таким чином, щоб можна було протестувати межі цього класу;

2) при розробці тестів розглядаються як вхідні умови, так і множини результатів (вихідні класи еквівалентності).

Метод функціональних діаграм характеризується набором алгоритмічних правил для вибору значень операндів в операторах програми з визначенням очікуваних вихідних значень для кожного застосовуваного тесту і є досить громіздким в реалізації.

При застосуванні методу припущення про помилку складається список можливих помилок, а також особливих ситуацій, в яких імовірно можуть з'явитися помилки. Потім, використовуючи цей список, складаються відповідні ТНД. У визначенні можливих помилок може послужити поглиблений аналіз специфікацій на програму (визначаються моменти, не описані в специфікаціях).

Основним змістом методів динамічного тестування є виконання тестів програмами, приведеними до виду завантажувальних модулів. Серед цих методів особливо складним представляється метод детермінованого динамічного тестування, при якому контролюються всі можливі комбінації вихідних даних, що дозволяє виявити відхилення у результатах роботи програми від еталонних значень, позначених у специфікаціях. Але на практиці, вже в програмах середньої складності перебрати всі комбінації вхідних даних виявляється неможливим. У зв'язку з цим широке поширення набуло стохастичне динамічне тестування, при якому тестові дані генеруються у вигляді множин випадкових величин із заданими розподілами, що дає можливість варіювати вихідними даними. Недолік цього методу полягає в тому, що окремі помилки можуть бути не виявлені, якщо вони не виходять за рамки середніх статистичних оцінок.

Тестування програм по стратегії "білого ящика" ведеться з використанням методик, які покривають логіку (текст) програми, причому чим вище ступінь покриття логіки програми, тим більше всебічними вважаються ТНД.

Найбільш формалізованим методом перевірки програмного забезпечення є статичне тестування. При використанні цього методу в якості еталону беруться правила структурної побудови програмних модулів, а перевірка виконання цих правил проводиться за допомогою аналізу тексту програм на мові програмування. Звідси даний метод отримав свою другу назву - символічне тестування. Метод символічного тестування може бути заглиблений аж до рівня формальної верифікації та доказу правильності програм. Такий підхід до тестування програмних модулів забезпечує всебічну перевірку ПЗ, але надзвичайно трудомісткий і вимагає великих матеріальних витрат.

Результати порівняльного аналізу методів із стратегій "білого" і "чорного" ящиків з метою визначення переважних методів при тестуванні програмних модулів контролю польоту наведені в таблиці 2.10.

Таблиця 2.10. Порівняльний аналіз методів тестування

Методи тестування	**Модулі комплексу відтворення**	**Модулі комплексу допускового контролю**	**Модулі комплексу контролю якості польоту**
Еквівалентне розбиття	+	+	+
Аналіз граничних значень		+	+
Функціональні діаграми			
Припущення про помилку	+		+
Динамічне тестування	+	+	+
Стохастичне тестування		+	+
Покриття операторів	+	+	+
Покриття рішень/умов		+	+
Комбінаторне покриття умов	+	+	+
Статичне тестування	+	+	+
Доказ правильності програм	+	+	+

Оскільки модулі відтворення в основному займаються візуалізацією ПІ, при їх тестуванні в меншій мірі застосовні методи аналізу граничних значень і стохастичного тестування, які добре "працюють" для модулів з комплексів допускового контролю та контролю якості польоту, так як ці комплекси повинні якісно визначати події контролю. З причини того, що модулі допускового контролю реалізують алгоритми контролю пілотування і працездатності НД, при їх тестуванні мало що може дати метод припущення про помилку, який може бути вдало застосований при тестуванні модулів з комплексу відтворення.

2.5.3. Комплексна стратегія проектування тестів для програмних модулів обробки польотної інформації

Оскільки застосування методів із стратегій "білого" і "чорного" ящиків забезпечує створення набору тестів, що покривають кожний свою область вхідних і вихідних значень і логіки модуля, пропонується визначити комплексну стратегію, яка буде складатися з наступних правил:

1) Використовуючи метод еквівалентного роздроблення, виділити правильні і неправильні класи еквівалентності за вхідним і вихідним даним. Метод еквівалентного роздроблення враховує синтаксичні та семантичні правила побудови програм, і способи завдання даних, і тому він ефективний для тестування будь-яких модулів і програм, що входять до складу АСКП.

2) Обов'язково використовувати метод граничних значень, включаючи аналіз граничних значень для вхідних і вихідних змінних програми. Аналіз граничних значень може дати набори додаткових тестових умов, тому необхідно розглянути тести, побудовані на попередніх кроках і доповнити їх.

3) Логіка програми перевіряється згідно критеріїв покриття рішень/умов або комбінаторного покриття умов, з яких останній є найбільш повним. Рекомендується доповнювати набір тестів знову побудованими тестами, що задовольняють одному з наведених критеріїв покриття.

4) Додаткові тестові набори даних можна отримати, використовуючи метод припущення про помилку.

Запропонована комплексна стратегія тестування дасть можливість провести більш повне й ефективне тестування програмних продуктів контролю польотів і забезпечити виявлення максимальної кількості помилок у тестованих модулях в порівнянні з використанням кожного з методів окремо.

2.5.4. Тестування функціональних груп модулів

Перш ніж тестувати комплекс програм КП цілком, необхідно протестувати складові його компоненти - модулі, що містять, як правило, невелика кількість операторів, а потім провести тестування групи модулів, що складають програму, підсистему або систему.

При аналізі підходів до тестування модулів виникає дилема: тестувати Чи кожен модуль окремо, а потім збирати програму і тестувати її в цілому, або модулі для проходження тестування послідовно підключати до групи раніше протестованих модулів. Перший підхід носить назву монолітного методу тестування, а другий - покрокового методу тестування та складання програм. Порівняльний аналіз цих методів дозволяє зробити наступні висновки:

1) При покроковому тестуванні можна ефективніше протестувати міжмодульних інтерфейси і виявити помилки в них на ранній стадії тестування з огляду на те, що збірка програми починається з підключення до групі тестованих модулів другого модуля. На противагу покроковому методу, при монолітному тестуванні модулі збираються в єдину програму на останньому етапі тестування, і тому налагодження міжмодульних інтерфейсів перевантажена, тому доводиться тестувати одразу всі можливі зв'язки між модулями.

2) При застосуванні монолітного методу тестування неминучі великі витрати праці, ніж при застосуванні покрокового методу.

3) Результати покрокового тестування якісніші, оскільки тестування модулів відбувається комплексно при покроковому нарощуванні кількості модулів. При монолітному тестуванні такі можливості відсутні.

4) При застосуванні покрокового методу раніше визначаються помилки в інтерфейсах між модулями, тому налагодження програм більш ефективна.

5) Використання обчислювальної техніки при монолітному тестуванні ефективніше в силу того, що тестовані ланцюжка модулів довше при використанні покрокового методу і коротше при монолітному методі.

6) Застосування монолітного тестування надає великі можливості для паралельного виконання робіт на першому етапі.

Переваги покрокового підходу демонструються в пунктах 1), 2), 3), 4). Оскільки наслідки невиявлених помилок в АСКП можуть бути дуже серйозні, ці переваги перекривають недоліки, відображені в пунктах 5) і 6).

Проаналізовано практичні прийоми проведення тестових випробувань функціональних груп модулів. Проведений аналіз дозволяє зробити висновок: застосування покрокового методу тестування при тестуванні ПЗ КП переважніше методу монолітного тестування, оскільки результати покрокового тестування якісніші.

Переваги покрокового підходу демонструються в пунктах 1), 2), 3), 4). Оскільки наслідки невиявлених помилок в АСКП можуть бути дуже серйозні, ці переваги перекривають недоліки, відображені в пунктах 5) і 6).

Проаналізовано практичні прийоми проведення тестових випробувань функціональних груп модулів. Проведений аналіз дозволяє зробити висновок: застосування покрокового методу тестування при тестуванні ПЗ КП переважніше методу монолітного тестування, оскільки результати покрокового тестування якісніші.

Покроковий метод тестування модулів складається з низхідній і висхідній стратегії. Яка з цих двох стратегій може забезпечити більш ефективне тестування програм у програмних комплексах контролю польотів?

Розгляд цих стратегій дозволив визначити наступні переваги висхідній стратегії тестування модулів:

1) можливість ранньої локалізації помилок, якщо помилки знаходяться в основному в модулях нижнього рівня;

2) простіше процес створення тестових умов і оцінка результатів тестування.

Недоліки висхідній стратегії полягають у наступному:

1) програма як ціле не існує, поки до групи тестованих модулів не буде доданий останній модуль;

2) необхідна розробка програм-драйверів.

У свою чергу переваги низхідного тестування модулів виражаються в наступному:

1) можливість виявлення помилок на ранній стадії тестування, якщо помилки знаходяться в основному у верхніх модулях програми;

2) структура програми формується на ранніх етапах тестування.

До недоліків низхідного тестування можна віднести наступні:

1) необхідно розробляти модулі-заглушки, що є непростим завданням;

2) складно забезпечувати тестовими даними тестовані модулі за допомогою зчитування їх через модулі-заглушки;

3) важко створювати тестові умови і складно оцінювати результати тестування, оскільки роботу модулів нижніх рівнів повинні імітувати заглушки;

4) стимулюється не завершення тестування через складність побудови різних версій коректних заглушок, що забезпечують надходження ТНД.

З урахуванням пунктів 2) і 3) з недоліків низхідного тестування та пункту 2) з переваг висхідного тестування перевага віддається стратегії висхідного тестування.

За результатами оцінки методів тестування функціональних груп модулів зроблені висновки про можливість застосування цих методів при тестуванні ПЗ КП, які наведені в таблиці 2.11

Таблиця 2.11. Оцінка методів тестування функціональних груп модулів

Методи тестування	**Програми комплексу відтворення**	**Програми комплексу допускового контролю**	**Програми комплексу контролю якості польоту**
Покроковий метод	+	+	+
Метод монолітного тестування			
Спадне тестування			
Висхідне тестування	+	+	+

Наведені методи можна ефективно застосовувати для тестування наступних компонент: підпрограм, що входять до складу програм обробки ПІ; програм, що входять до складу систем або комплексів КП.

2.6. Методи створення ТНД при сертифікаційних випробуваннях комплексів програм контролю польотів

2.6.1. Тестування програмних комплексів контролю польотів

Програмне забезпечення контролю польотів включає в себе три основних програмних комплексу:

- Комплекс програм відтворення польотної інформації;
- Комплекс програм допускового контролю польоту;
- Комплекс програм контролю якості виконання польоту.

Хоча вхідний інформацією для кожного з комплексів контролю польотів служить

інформація параметричних бортових реєстраторів, проте в кожному з них реалізовані різні класи алгоритмів і різні принципи організації програм. Цей факт визначає різні вимоги до методів тестування та використовуваним ТНД.

Комплекс програм відтворення вирішує задачу перетворення ПІ з видачею графіків або цифрових значень на екран дисплея або друк. У комплексі програм відтворення реалізовані алгоритми попередньої обробки інформації та перетворення кодових значень параметрів у фізичні. Основною вимогою до виконуваних функцій є вимога забезпечення заданої точності відтворення.

Суттєвою особливістю тестування цього комплексу є те, що реалізовується набір алгоритмів при кожному тестовому проході не залежить від вхідних даних, а послідовність виконання алгоритмів є детермінованою. Тестовими даними для перевірки такого комплексу можуть бути довільні послідовності значень із припустимої області. У цьому випадку для завдання ТНД зручно використовувати значення функцій, які можна легко вирахувати. Крім того, для тестування цього комплексу рекомендується використовувати тестову копію польоту з повним описом зареєстрованої інформації.

Основним завданням, розв'язуваної при тестуванні модулів відтворення, є оцінка точності виконуваних перетворень, таких як фільтрація випадкових похибок, інтерполяція, аналогово-цифрові перетворення та ін. Тому для тестування програм комплексу відтворення можна ефективно застосувати методи динамічного тестування, а також методи еквівалентного роздроблення, аналізу Граничне значення і припущення про помилку.

Комплекс програм допускового контролю повинен забезпечувати реалізацію алгоритмів контролю, визначених в експлуатаційній документації на ЛА. Комплекс допускового контролю призначений для виявлення відхилень у роботі систем НД і помилок пілотування, які можуть впливати на безпеку польоту. Тому він повинен забезпечувати рішення наступних завдань:

• ідентифікація етапів польоту та визначення подій контролю;

• ідентифікацію режимів роботи систем ЛА;

• визначення виходу контрольованих параметрів за обмеження.

У програмному комплексі допускового контролю реалізовані алгоритми обчислення складних логічних функцій, визначених на множині безперервних взаємопов'язаних параметрів, які описують траєкторію руху динамічної системи. Характерною особливістю цього комплексу є залежність набору алгоритмів і послідовності їх реалізації від вхідних даних. Тому при тестуванні комплексу необхідно забезпечити повноту ТНД, які повинні покривати всі алгоритми контролю. Для створення ТНД при тестуванні комплексу допускового контролю доцільно використовувати комплексні імітаційні стенди, де за заздалегідь розробленим сценарієм проводиться корекція інформації штатного польоту з

метою моделювання контрольованих порушень і виходів за обмеження. Завдання конкретних значень параметрів в моделі повинно здійснюватися з урахуванням забезпечення необхідної достовірності об'єктів контролю.

Комплекс програм контролю якості польоту здійснює контроль за виконанням екіпажами ЛА режимів і правил льотної експлуатації. Для вирішення завдань контролю якості виконання польоту ПЗ комплексу повинне виконувати наступні функції:

• формування інформаційного "портрета" етапів польоту;

• визначення значень показників якості пілотування.

Комплекс контролю якості польоту включає в себе алгоритми характерні як для комплексу відтворення, так і для комплексу допускового контролю. Комплекс реалізує алгоритми пошуку контрольованих ситуацій в копії польоту, а також алгоритми відтворення значень параметрів в контрольованих точках польоту або на його ділянках. Крім того, тут реалізовані алгоритми функцій, що визначають значення параметрів пілотування. В якості ТНД можна використовувати тестові дані для комплексу відтворення, а також ПІ штатного польоту, яку при перевірці показників якості пілотування необхідно відкоригувати з метою внесення виходів за обмеження. Така корекція може бути проведена з використанням імітаційного моделювання. Кращі методи тестування комплексів програм контролю польотів наведено в табл. 2.12.

Таблиця 2.12. Методи тестування комплексів програм контролю польотів

Методи тестування комплексів програм	**Комплекс програм відтворення**	**Комплекс програм допускового контролю**	**Комплекс програм контролю якості польоту**
Методи динамічного тестування	+	+	+
Метод імітаційного моделювання	-	+	+

2.6.2. Етапи оціночного тестування комплексів програм контролю польотів

Тестування комплексів програм контролю польотів має свою специфіку і відрізняється від простого тестування модулів зважаючи на складність і різноманіття зв'язків між модулями, програмами та системами, що входять до складу кожного комплексу. Виділимо загальні категорії тестів, які можуть бути застосовані для оціночного тестування комплексів програм КП.

При тестових випробуваннях комплексів КП на першому етапі слід перевірити даний комплекс на відповідність його специфікаціям, які повинні містити точний опис поведінки програм, що складають комплекс, з точки зору "зовнішнього світу". Специфікації на кожен

програмний модуль повинні містити описи, типи та характеристики наступних компонент:

- Зовнішні змінні (глобальні);
- Внутрішні змінні (локальні);
- Вхідна предметна область (вхідні дані);
- Вихідні дані (межі очікуваних інтервалів);
- Викликаються модулі (очікувані результати).

Мета тестування специфікацій полягає в перевірці відповідності результатів вихідним даним, описаним у специфікаціях. Якщо інформація на входах і виходах програмних модулів, взаємодіючих між собою в складі комплексу програм, однозначно відповідає один одному згідно своїм специфікаціям, то вважається, що такі програми задовольняють вимогам специфікацій.

В якості другого етапу при тестуванні комплексів програм контролю польотів виступає оцінка повноти переліку функцій, виконуваних комплексом, згідно специфікаціям. Для визначення повноти переліку функцій необхідно скласти тестові набори даних по кожній з розв'язуваних комплексом завдань, забезпечити проходження кожного тесту і оцінити результати тестування. У разі задовільної оцінки комплекс програм вважається функціонально повним.

Третім етапом є тестування програмних модулів. Основною метою тестування модулів є перевірка якості обробки інформації, що надходить на вхід модулів, для чого здійснюється оцінка інформації на їх виході. Крім того, при тестуванні модулів, логіка кожної програми перевіряється по методу комбінаторного покриття умов, а оцінка вхідним і вихідним класам еквівалентності дається згідно методу аналізу граничних значень для кожного модуля.

Четвертим етапом є тестування функціональних груп програм (підсистем), яке проводиться з метою перевірки інформаційних і керуючих зв'язків між модулями, програмами і підсистемами, що входять до складу комплексу КП. Тут проводиться тестування міжмодульних, а також людино-машинних інтерфейсів. Для тестування структури підсистем та шляхів проходження інформації рекомендується скористатися детермінованим динамічним тестуванням з застосуванням стохастичного підходу. Коректність функціонування підсистем контролюється шляхом проходження вихідних еталонних даних і оцінки відхилення результатів від еталонних значень.

П'ятий етап полягає в тестуванні комплексу програм КП в цілому, при якому здійснюються наступні дії:

• оцінка комплексу програм на відповідність технічній документації;

• оцінка поведінки підсистем в критичних умовах на граничних (критичних) наборах даних;

• оцінка витрат ресурсів, наявність захисту даних і стійкість комплексу до збоїв апаратури і ОС;

• контролюється структурна схема комплексу програм, яка визначає логіку взаємодії програм, а також інформаційні потоки усередині комплексу.

2.6.3. Динамічне тестування комплексів програм контролю польотів в реальному часі та імітаційне моделювання

Тестування будь-якого комплексу програм у реальному часі являє собою виконання програм, що входять до його складу, з урахуванням часу проходження тих чи інших груп тестів, тривалості обробки, взаємодії з іншими програмами та модулями. У разі неспівпадіння результатів з еталонними, фіксується час настання помилки, визначається модуль її викликав, і далі необхідно перейти до детермінованому тестуванню цього модуля для локалізації та виправлення помилки.

При тестуванні комплексів програм контролю польотів потрібно задавати велику кількість вихідних даних складної структури, а також еталонних вихідних значень для оцінки результатів тестування. Оскільки створення такого обсягу даних вручну практично неможливо, для генерації ТНД і еталонних значень зазвичай використовують програмні моделі й імітатори. Програмні моделі являють собою програмні продукти, які забезпечують створення тестів, що покривають множину допустимих вхідних даних, що описують поведінку об'єктів у зовнішньому середовищі.

Генератори ТНД і імітатори зовнішнього середовища можна використовувати роздільно з тестованим комплексом програм, заздалегідь готуючи і записуючи ТНД на магнітні носії, звідки ці набори, що синхронізуються відповідно з реальним часом, будуть надходити на вхід тестованого комплексу. У процесі всеосяжного тестування комплексів програм особливе значення набуває генерація і введення тестових наборів даних у темпі реального часу. Для вирішення цього завдання потрібна співвіднести в часі наступні процеси:

1) генерація вхідних ТНД;

2) функціонування тестованого комплексу програм;

3) оцінка результатів тестування поточної підсистеми або програми.

Динамічне тестування комплексів програм у реальному часі є складним завданням, а імітація зовнішнього середовища і генерація тестової інформації в процесі динамічного тестування може бути ефективно забезпечена тільки за умови максимальної автоматизації цього процесу.

Основними характеристиками динамічного тестування є наступні:

1) Більшість знов генеруються тестів може залежати від попередніх порцій вихідної інформації тестованого комплексу. Такі тести повинні генеруватися за мінімальні часові

відрізки, щоб бути введеними в комплекс в темпі реального часу.

2) Багато ТНД містять логічні величини, пов'язані з іншими змінними, в тому числі з цілими, змінними з плаваючою точкою, а також булівськими змінними. Такими залежностями пов'язані змінні, які описують поведінку фізичних об'єктів у зовнішньому середовищі. У цьому випадку завдання полягає в коректній імітації всіх взаємопов'язаних між собою змінних при створенні ТНД.

3) Слід забезпечити генерацію таких ТНД, які включають в себе модифіковані еталонні набори вхідних і очікувані набори вихідних даних з характеристиками внесених спотворень за вхідними даними.

4) Необхідно забезпечити повторюваність ТНД при виявленні відхилень у поведінці комплексу з метою локалізації помилок. Проблема тут полягає у великій кількості ТНД, які генеруються за час випробувань комплексу.

Масштабні завдання такого роду можуть бути вирішені із залученням потужних обчислювальних систем. Як правило, для створення ТНД виділяється спеціальний технологічний комп'ютер, на якому встановлено ПЗ генерації ТНД і обробки результатів випробувань. Технологічний комп'ютер з'єднується за допомогою засобів комутації з комп'ютером на якому функціонує тестований комплекс програм. Такі обчислювальні установки називаються імітаційно-моделюючими стендами (рисунок 2.5).

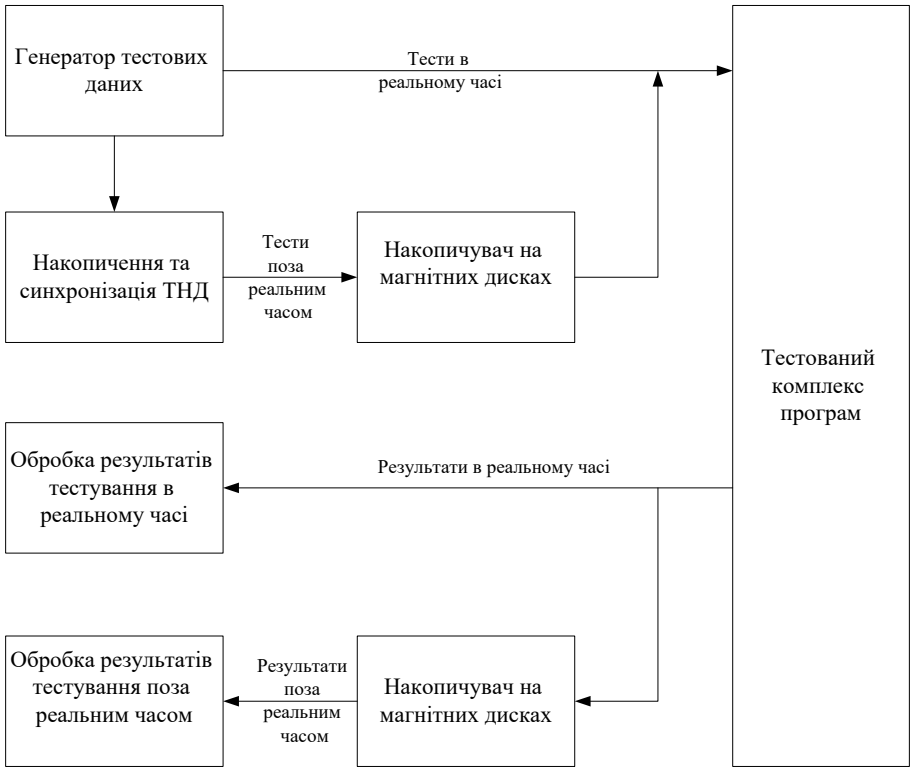

Рис. 2.5. Функціональна організація процесу тестування за допомогою імітаційно-моделюючого стенда

Імітаційно-моделюючі стенди можна застосовувати для тестування комплексів і систем обробки польотної інформації, записаної бортовими реєстраторами. Зауважимо, що цей процес вимагає великих матеріальних вкладень, так як для імітації реальної ПІ необхідна розробка спеціальної апаратури.

2.6.4. Технологія створення тестових наборів даних при сертифікаційних випробуваннях комплексів програм контролю польотів.

Під час тестування комплексів програм контролю польотів можна успішно застосувати принципи динамічного тестування в реальному часі, викладені у Розділі 3. Для наземних АСКП фактор реального часу не є визначальним, оскільки інформацію бортових засобів реєстрації польоту можна обробляти поза реальним часом. Суттєвим тут є вид тестової інформації, яка повинна імітувати реальну ПІ, що відображає поведінку ЛА.

При тестуванні модулів відтворення для окремих АП та РК можна використовувати заздалегідь підготовлені еталонні вибірки вхідних величин з відомою формою графічного представлення. Однак, при тестуванні модулів допускового контролю, де оцінюються групи взаємопов'язаних фізичних параметрів в динаміці їх зміни, найрентабельнішим рішенням для створення ТНД є використання реальної польотної інформації.

Виготовлення апаратури, що імітує ПІ, наприклад, у вигляді набору датчиків-імітаторів, керованих комп'ютером, є непростим завданням. Тому для імітації ПІ пропонується використовувати штатні копії польотів, записані бортовими засобами реєстрації під час польоту ЛА. Таким чином, на базі технологічного комп'ютера пропонується побудувати програмний імітаційно-моделюючі стенд без використання спеціальної апаратури.

У відповідності з цим підходом побудований програмний комплекс створення ТНД і обробки результатів тестування, який входить в якості основної компоненти в систему сертифікаційних випробувань ПЗ КП (рисунок 2.6).

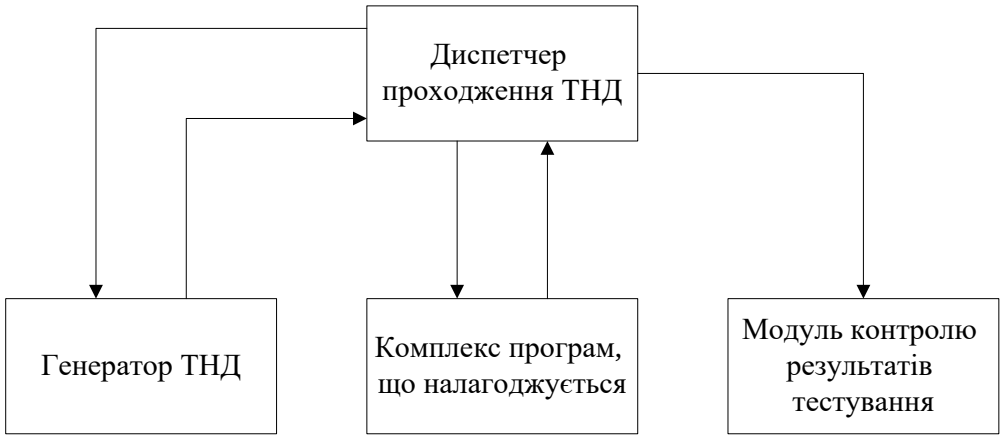

Рис.2.6. Функціональна організація процесу тестування в комплексі створення ТНД і обробки результатів тестування

Програмний комплекс створення ТНД і обробки результатів тестування базується на наступних принципах:

1) Для побудови тестових наборів даних у реальних копіях польотів моделюються контрольовані події, а також вносяться спотворення (викиди, шуми і т.п.).

2) Події контролю вносяться в копію польоту спеціально розробленими модулями, причому величини відхилень контрольованих фізичних параметрів визначаються з урахуванням фактора вірогідності контролю.

3) Модулі, що визначають наступ подій контролю, будуються із застосуванням логіки предикатів, рівнянь кінцевих автоматів і апарату магазинних граматик.

4) В процесі підготовки ТНД надаються можливості по корекції копій польотів записаних різними типами БР. Реалізовані функції редагування аналогових параметрів, разових команд, часу і розпізнавальних даних (номер борта, номер рейсу, дата польоту і т.п.).

Запропонований підхід до створення ТНД назвемо методом імітаційного моделюванн. Наведемо основні правила, що визначають цей метод:

1) Для створення ТНД використовується технологічний комп'ютер зі спеціальним ПЗ - програмним комплексом генерації ТНД і обробки результатів випробувань.

2) Оскільки наземна обробка ПІ може проводитися поза реального часу, тестування комплексів програм обробки ПІ проводиться також поза реального часу з використанням принципів динамічного тестування.

3) Предметною областю тестованих АСКП і іншого ПЗ КП є інформація, записана БР в реальному часі на борту ЛА під час польоту.

4) Тестові набори даних повинні імітувати копії польотів ЛА. Така імітація зовнішнього середовища досягається за допомогою застосування як ТНД відредагованих штатних копій польотів ЛА з розпізнавальними даними і тимчасовими позначками.

5) Програмний комплекс створення ТНД забезпечує редагування копій польотів різних типів НД, записаних різними марками БР.

6) Виходи за обмеження моделюються в штатних копіях польотів за допомогою програмного комплексу створення ТНД, який використовується також і для оцінки результатів випробувань ПЗ КП.

7) Оцінка результатів тестових випробувань ПЗ КП проводиться на технологічному комп'ютері за результатами випробувань комплексу відтворення, визначення достовірності об'єктів контролю в комплексі допускового контролю, якості попередньої обробки та іншими показниками.

Розділ 3. Використання технології тестування критичних програмних систем

3.1. Створення групи тестування

Створення групи тестування – перший, але дуже важливий крок в управлінні процесом тестування, на якому визначається її рівень і підпорядкованість, порядок взаємодії з розробниками і замовниками.

Завдання 1.1. Визначення сфери діяльності групи тестування. Рівень групи:

- організація. Бере участь в тестуванні всіх проектів організації і підпорядковується керівникові організації;
- відділ (підрозділ). Бере участь в тестуванні проектів, що розробляються підрозділом, і підкоряється керівникові відділу;
- проект. Бере участь в тестуванні певного проекту внутрішня група, що входить в групу розробки і підкоряється керівникові (менеджерові) проекту.

Рівні тестування: автономне, інтеграційне, тестування ПЗ, системне тестування. Відповідальності на всіх рівнях.

Потреба в інструментах тестування. Їх вибір, придбання і освоєння, або розробка власних інструментів.

Завдання 1.2. Визначення учасників процесу тестування.

Визначити коло осіб, які братимуть участь в процесі тестування. Склад і кількість учасників процесу тестування визначається наявними ресурсами (трудовими і матеріальними), об'ємом, складністю і критичністю проекту.

В процесі тестування беруть участь: представники замовника і користувачі; керівник проекту; група якості; група тестування; розробники (аналітики і програмісти); адміністратор бази даних.

Користувачі притягуються до процесу тестування для: узгодження серйозності відмов і дефектів; реєстрації відмов і дефектів при дослідній експлуатації; участі в підготовці планів приймального тестування; оцінки зручності вживання і адекватності підтримки вирішення завдань в ПЗ.

Завдання 1.3. Розподіл обов'язків і формування плану роботи групи тестування.

Зразковий розподіл обов'язків в рамках кроків процесу вже був представлений в таблиці 3.1. Розподіл завдань тестування між членами групи визначається чисельним складом групи. Наприклад, в крупних проектах розробку планів, сценаріїв, проектів тестів здійснює аналітик («архітектор тестів»), а виконання тестів – рядові тестери. У невеликих проектах всю роботу може виконувати навіть одна людина.

Після створення групи тестування і розподілу обов'язків, розробляється план роботи групи тестування, в якому відбиваються перелік завдань тестування, які мають бути виконані, відповідальні виконавці за кожне завдання і орієнтовні терміни початку і завершення (погоджені з планом робіт за проектом).

Вимоги до кваліфікації тестувальників. У багатьох організаціях на роль тестерів запрошуються фахівці з нижчою кваліфікацією, ніж програмісти. Проте тестування вимагає від людини як широкого кругозору, так і певних ділових якостей. Ось. наприклад, перелік якостей, котрим повинен володіти ідеальний тестувальник:

- уміння руйнувати програмні продукти;
- уміння розробляти і виконувати сценарії і процедури тестування;
- уміння описувати послідовність подій і конфігурацію системи, які призводять до виникнення проблеми (наприклад, чітко документувати процедури і результати, уміння усно передавати інформацію розробникам, іншим тестувальникам і керівництву);
- здатність критикувати і коректно сприймати критику;
- уміння протистояти тиску (тестування завжди є завершальною стадією будь-якого процесу розробки, і як правило, проходить в стресових умовах);
- здатність швидко освоювати нові технології;
- терпіння. Потрібно бути готовим виконувати прогони тестів стільки разів, скільки потрібно для того, щоб виявити проблему, після чого повторно виконати тести, аби переконатися в її коректному усуненні.
- гнучкість мислення – здатність швидко перемкнутися на тестування нового програмного продукту або навіть відмовитися від тестування одного продукту на користь іншого, що володіє вищим пріоритетом.
- широта мислення – здатність одночасно бачити загальну картину і вміння зосередитися на деталях.
- бути експертом в декількох областях – групі тестування можуть знадобитися фахівці з баз даних, по комунікаціях, по мережевих технологіям, по тестуванню призначених для користувача інтерфейсів, а також фахівці з інших областей.

Взаємозв’язок робочих продктів процесу тестування зображений на рисунку 3.1.

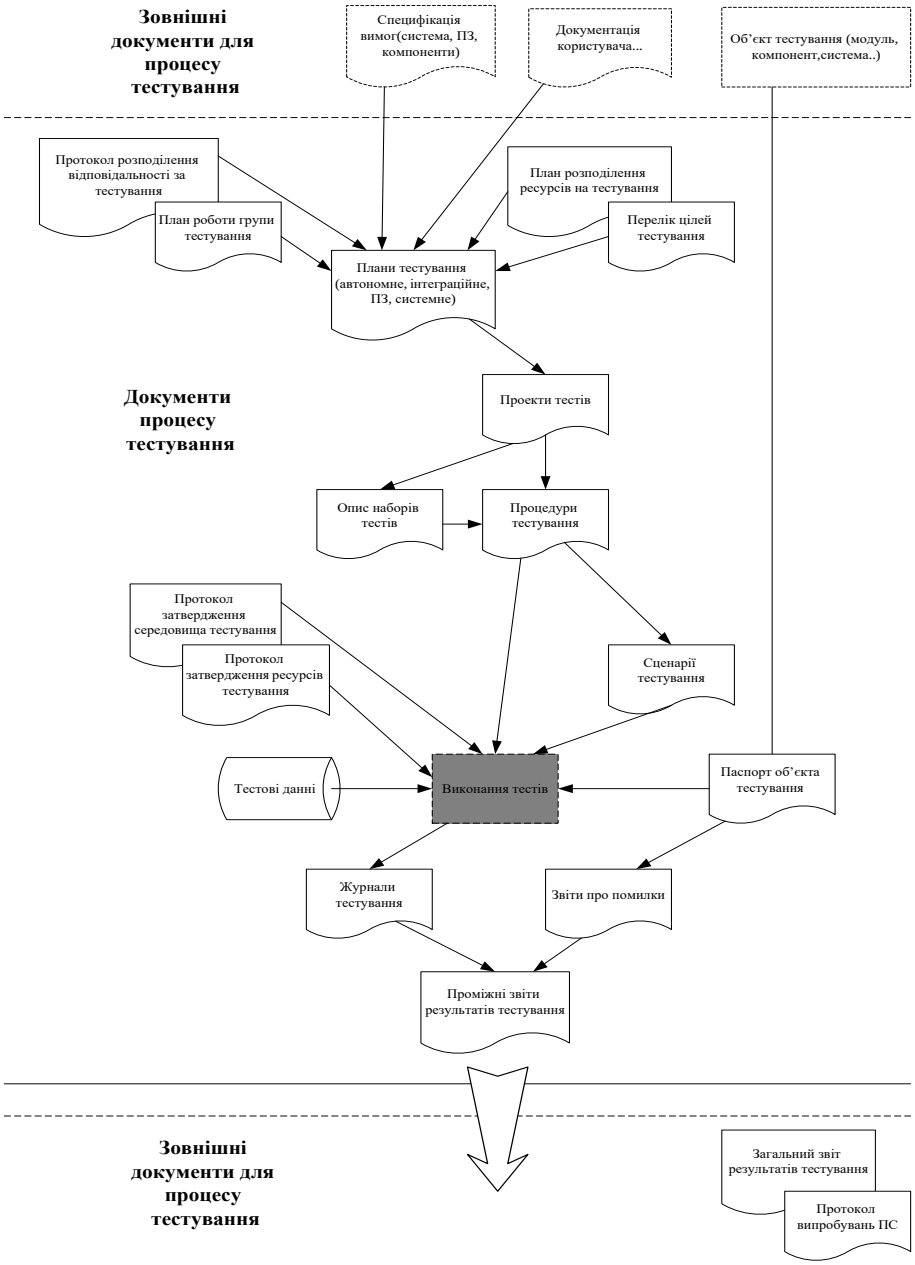

Рис. 3.1. Взаємозв'язок робочих продуктів процесу тестування

В таблиці 3.1 представлений приблизний розподіл обов'язків в рамках кроків процесу.

Таблиця 3.1. Розподіл обов'язків по виконанню процесу тестування

Крок процесу тестування	Виконавець
Створення групи тестування	Керівник проекту(менеджер)
Аналіз ризику	Керівник проекту(менеджер), керівник групи тестування, аналітики проекту, група якості
Визначення мети тестування	Керівник групи тестування, аналітики проекту, група якості
Розробка планів тестування	Тестувальники
Розробка тестів	Розробники, тестувальники
Автономне та інтеграційне тестування	Розробники, тестувальники
Тестування ПЗ	Тестувальники
Системне тестування	Тестувальники
Аналіз результатів тестування	Керівник групи тестування, тестувальники, група якості
Регресійне тестування	Тестувальники

3.2. Аналіз ризиків

Процесу тестування, головним чином, стосуються ризики, зв'язані з технічними аспектами розробки, середовищем і технологією розробки, а також питання управління проектом. Результати аналізу ризиків використовуються для вироблення загальної стратегії тестування, визначення стану тестування шляхом відстежування відмов і дефектів і визначення критеріїв завершення задач тестування.

Завдання 2.1. Ідентифікація ризику.

Рішення задачі полягає в розгляді і аналізі функціональних і нефункціональних вимог до ПС, характеристик її розміру і складності, а також середовища і умов використання, і виявленні чинників можливих ризиків для процесу тестування і майбутнього функціонування ПС. Окрім використання загальних питань, пропонованих в таксономії ризиків, при ідентифікації ризиків група тестування повинна враховувати ризики, зв’язані з якістю ПС, зокрема, по наступним атрибутам:

Функціональність.

Надійність.

Зручність вживання.

Безпека.

Для виявлення потенційних ризиків, пов'язаних з об'ємом і складністю тестування, розглядаються питання, що стосуються характеристик ПС і умов її експлуатації, наприклад:

Складність ПС.

Умови експлуатації.

Характеристики середовища.

Крім того, при аналізі ризиків розглядаються питання, що стосуються ефективності процесу тестування, наприклад:

Вимоги і специфікації системи і ПЗ

Модель розробки.

Перевірки.

Терміни.

Середовище тестування.

Процес тестування.

В результаті розгляду питань, визначуваних таксономією ризиків, формується список ризиків, які можуть перерости в проблеми і привести до зриву процесу тестування і випуску програмного продукту з серйозними дефектами. Наступний крок в ідентифікації ризиків - визначити вірогідність і серйозність кожної потенційної проблеми для процесу тестування.

Завдання 2.2. Ідентифікація програмних компонентів ПС, які тестуватимуться.

Отриманий список доповнюється цілями тестування нефункціональних характеристик, співвіднесених з компонентами ПС. Ідентифікація цілей виконується на основі аналізу відповідних документів системи. Якщо розробка виконується за допомогою CASE-засобів - з опису проекту в їх середовищі.

Завдання 2.3. Визначення критеріїв проходження тестів.

Після побудови списку цілей потрібно визначити критерій ухвалення рішення про досягнення кожної з них (як взнати, що мета досягнута?). Критерії можуть бути якісними і кількісними. Якісні критерії формуються по очікуваному результату виконання тестів (прохід/відмова). Кількісні критерії ґрунтуються на метриках покриття (наприклад, всі заплановані тести пройшли), кількості залишених дефектів і інтенсивності відмов.

Завдання 2.4. Пріоритезація цілей тестування.

Рішення задачі полягає у визначенні пріоритетів виконання тестів (високий, середній, низький), які встановлюються на основі аналізу ризиків проекту і ризиків відмови.

3.3. Розробка плану тестування

План тестування визначається в стандарті IEEE Std. 829:1998 як «документ, в якому визначені об'єм, підходи, ресурси і тривалість тестування. У нім вказуються тестовані об'єкти (елементи), характеристики, виконувані завдання тестування, відповідальні виконавці по кожній задачі, а також ризики, пов'язані з планом». Цей стандарт не регламентує ні рівні, ні об'єкти тестування на цих рівнях, використовуючи загальні терміни «тестовані елементи» стосовно будь-яких об'єктів (модулів, компонентів, підсистем або систем) і «характеристики» стосовно будь-яких тестованих характеристик, і може використовуватися для розробки планів на кожному рівні тестування.

Плани можуть уточнюватися по мірі розробки системи (наприклад, при уточнення вимог або термінів), але мають бути готові і перевірені перед початком виконання тестування на відповідному рівні.

Структура плану тестування, що рекомендується стандартом IEEE Std. 829, приведена нижче. Слід зазначити, що загальні завдання планування випробувань перераховані в ДСТУ 2853-94, проте структура плану не представлена.

- Ідентифікатор плану

Унікальний ідентифікатор, що привласнюється плану.

- Вступ

Вказують короткий опис об'єктів тестування і тестовані характеристики, а також рівень плану (автономне, інтеграційне і ін.). Тут же вказують посилання на документи за проектом і використовувані стандарти.

- Тестовані елементи

Перераховують об'єкти тестування з вказівкою їх версії і рівня тестування, характеристики носіїв, що містять об'єкти тестування, вимоги до апаратних засобів або вимоги до інсталяції об'єкту перед початком його тестування. Вказують посилання на необхідні документи (наприклад, специфікації вимог і проекту, посібник користувача, керівництво по інсталяції, керівництво по обслуговуванню), а також на будь-які звіти про проблеми, що стосуються тестованих об'єктів.

- Тестовані характеристики

Вказують всі тестовані характеристики і їх комбінації. На наступному кроці процесу тестування цей розділ доповнюється переліком проектів тестів, пов'язаних з кожною характеристикою і кожною комбінацією характеристик.

- Не тестовані характеристики

Вказують, що не буде тестуватися і чому.

- Підхід

Описують загальний підхід до тестування. Для кожної групи характеристик або їх комбінації вказують підхід, який повинен гарантувати адекватність тестування. Вказують основні методи, стратегії і інструменти, які планується застосовувати для виконання тестування на відповідному рівні. Вказують мінімально необхідну міру охопленняу елементів процесом тестування, критерії, які будуть використані для ухвалення рішення про адекватність тестування (функціональне покриття, допустима кількість дефектів, частота відмов або час безвідмовного функціонування). Крім того, вказують методи, вживані для трасування вимог. Ідентифікують істотні обмеження, такі як міра готовності елементу до тестування, ресурси і терміни.

- Критерії проходження тесту

Вказують критерії, які використовуватимуться для встановлення факту, - пройшов або не пройшов тестований об'єкт кожен тест.

- Критерії призупинення і відновлення тестування

Перераховують причини, по яких можуть бути припинені роботи по тестуванню, а також завдання, які необхідно повторити при відновленні тестування. Документування результатів тестування. Перераховують документи, вживані для документування результатів тестування. Набір документів, що рекомендується, включає: план тестування; проекти тестів; опис набору тестів; описи тестових процедур; паспорти об'єктів, переданих на тестування; журнали тестування; звіти про проблеми.

- Завдання тестування

Перераховують завдання процесу тестування, що вирішуються при підготовці і виконанні тестування, залежності між завданнями, вимоги до кваліфікації виконавців.

- Середовище тестування

Вказують потрібну апаратно-програмну конфігурацію і інструменти.

- Обов'язки

Відповідальні виконавці по кожному завданню тестування. Питання кадрів і їх підготовки. Вказують вимоги до кількісного складу учасників процесу тестування і рівню кваліфікації фахівців, а також необхідність в їх навчанні.

- Календарний план

Вказують орієнтовні дати початку і завершення завдань процесу тестування, а також те, які ресурси і коли мають бути виділені.

- Ризик і непередбачені обставини.

Вказуються можливі ризики, пов'язані з виконанням завдань процесу тестування, і заходи, що робляться, по їх зменшенню.

- Затвердження

Вказують прізвища і посади осіб, зобов'язаних затвердити план, і резервують місце для їх підписів і дат затвердження.

Стратегія тестування програмних модулів АСКП представлена на рисунку 3.2.

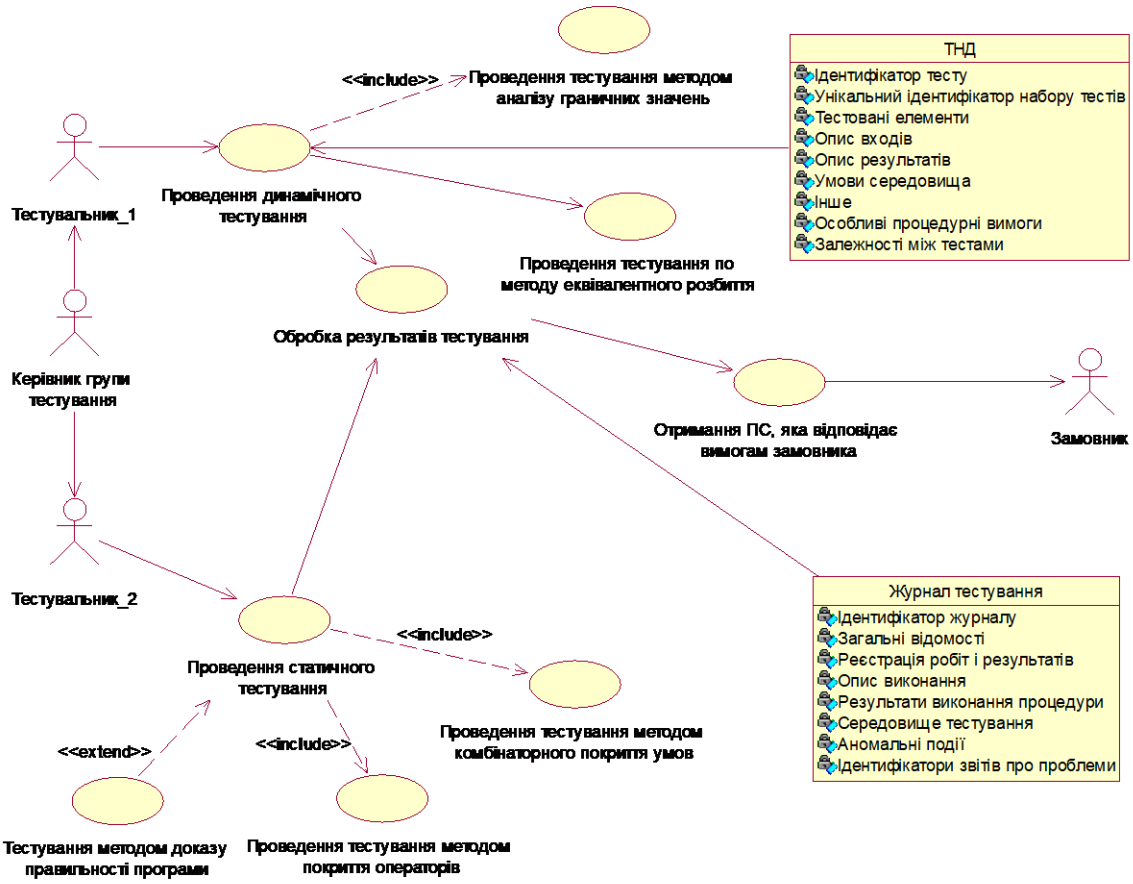

Рис. 3.2. Стратегія тестування програмних модулів АСКП

Завдання даного кроку виконуються на кожному рівні тестування. Вони описані нижче.

Завдання 3.1. Розробка плану приймальних випробувань системи.

Згідно стандарту ISO/IEC 12207 планування приймальних випробувань не пов'язане з процесом тестування, але група тестування може притягуватися для їх підготовки і проведення. Цей план повинен відображати інтереси різних користувачів системи; кінцевих користувачів, адміністраторів системи (мережі БД), персоналу супроводу (розвитку).

Завдання 3.2. Розробка плану тестування системи.

Цей план повинен охоплювати всі завдання тестування і розроблятися на основі вимог до системи. У нім перераховуються всі рівні і види тестів, і вказуються посилання на відповідні плани.

У плані мають бути вказані всі об'єкти тестування:

- всі зовнішні функції і компоненти ПЗ, впорядковані по пріоритетах тестування;
- необхідні конфігурації апаратних і програмних платформ: необхідні інтерфейси з іншими системами;
- документація споживача;
- учбові матеріали.

Всі плановані види тестування, в порядку пріоритету важливості: функціональне; надійності; продуктивності; конфігурації; безпеки; відновлення; регресійне. Для визначення тривалості і трудомісткості тестування можуть використовуватися методи оцінки розміру і складності.

Завдання 3.3. Розробка плану тестування ПЗ.

Цей план визначає стратегію перевірки функціональних характеристик ПЗ.

У плані мають бути відбиті цілі функціонального тестування ПЗ:

- всі зовнішні функції, впорядковані по пріоритетах тестування;
- опис інтерфейсу користувача (формати екранів, меню і так далі);
- характеристики продуктивності, розподілені по компонентам ПЗ;
- формати всіх типів даних і результатів (наприклад, всі звіти);
- документація користувача для перевірки її відповідності ПЗ;
- всі повідомлення про помилки, що формуються ПЗ;
- можливі області нестійкого функціонування (граничні значення);
- всі таблиці БД.

Завдання 3.4. Розробка плану інтеграційного тестування.

Цей план розробляється на основі плану інтеграції ПЗ і повинен відображати перелік компонентів ПЗ, порядок інтеграції і терміни виконання тестування. Планування тестування включає наступні дії:

ідентифікація інтерфейсів між компонентами ПЗ і порядку інтеграції;

ідентифікація необхідних інтерфейсів компонентів ПЗ з іншими компонентами системи (повторно-використовуваними компонентами, середовищем, устаткуванням);

ідентифікація інтерфейсів компонентів ПЗ з користувачем. Якщо компоненти мають ці інтерфейси (графічні інтерфейси), то в план має бути включене тестування їх узгодженості, а також вистави на різних типах моніторів і при різній роздільній здатності екрану.

Діаграма класів об’єктів тестування зображена на рисунку 3.3.

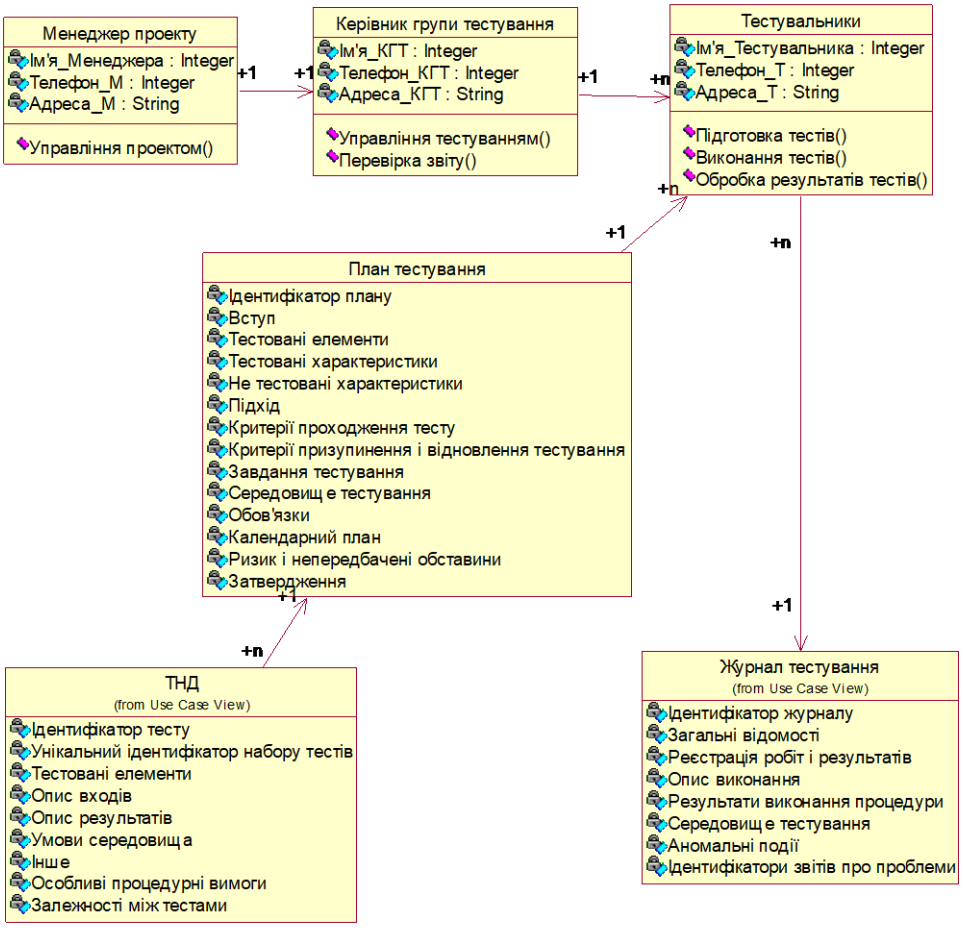

Рис. 3.3. Діаграма класів об’єктів тестування

Завдання 3.5. Розробка плану автономного тестування.

Планування автономного тестування включає наступні дії:

- ідентифікація функцій ПЗ, що реалізовуються в компоненті ПЗ;
- ідентифікація модулів, пов'язаних з реалізацією критичних функцій;
- ідентифікації складних і схильних до відмов модулів; визначення областей можливої риски відмови модулів;
- визначення джерел і форматів вхідних і вихідних даних (тестові дані, генератори даних);
- визначення граничних значень;

- визначення вимог до облиште тестування (по метриках структурного і функціонального покриття);
- визначення критерію завершення.

Цей план може бути розроблений у вигляді проекту тестів. Стратегія тестування підсистем програм контролю польотів представлена на рисунку 3.4.

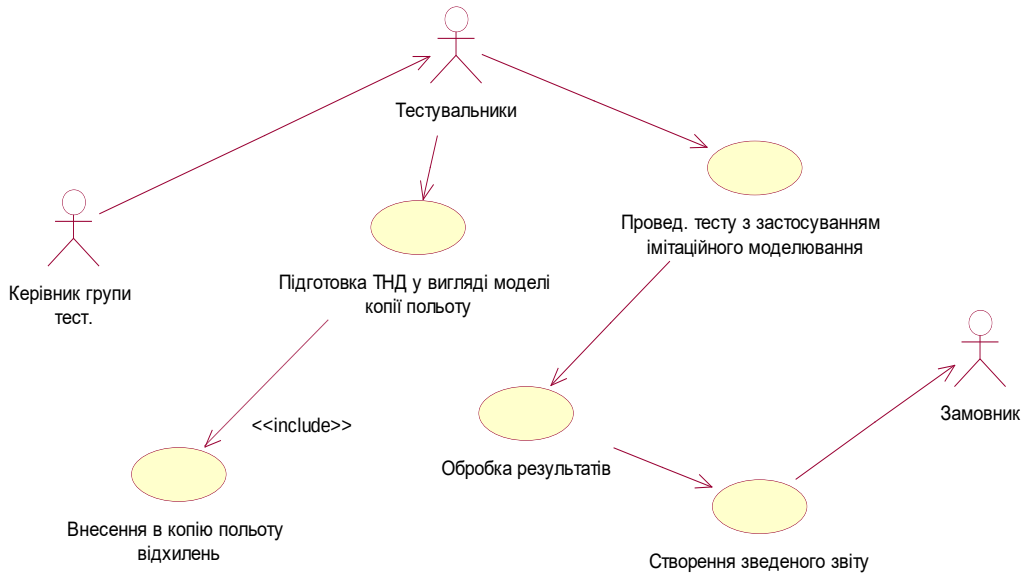

Рис. 3.4. Стратегія тестування підсистем програм контролю польотів

Завдання 3.6. Розробка плану регресійного тестування.

План розробляється на основі плану інтеграції ПЗ і повинен включати:

- перелік внесених змін;
- перелік тестів, які мають бути виконані повторно;
- тести для перевірки сумісності внесених змін з існуючими вимогами (нові тести);
- компоненти системи, які повинні тестуватися повторно (характеристики, функції, інтерфейси, компоненти, в яких були виявлені і усунені помилки).

У набір регресійних тестів потрібно відбирати невелику кількість раніше виконаних тестів: тести перевірки останніх змін в ПЗ, тести, при виконанні яких були виявлені серйозні дефекти, тести складних функцій і граничних умов.

Плани повинні піддаватися перевірці з метою з'ясування їх повноти і адекватності загальному плану розробки системи, плану якості, вихідним вимогам на розробку ПС. Затверджені плани передаються групі управління конфігурацією (окрім плану автономного тестування) для контролю внесення змін.

Крок виконується для кожного рівня тестування і відповідно до плану тестування на певному рівні. Вирішувані завдання перераховані нижче.

Завдання 3.7. Розробка тестів. Визначення підходів до розробки тестів.

Крок виконується для кожного рівня тестування і відповідно до плану тестування на певному рівні. Вирішувані завдання перераховані нижче. Рішення задачі полягає у виборі підходів/методів розробки тестів для кожного рівня тестування і виборі стратегії їх виконання. Визначення загального підходу. Ідентифікувати області ризику відмови, які будуть відстежуватися при тестуванні (таблиця ризиків відмов зовнішніх функцій). У таблиці 3.2 перераховані основні підходи/методи розробки тестів з вказівкою рівнів, на яких їх доцільно застосовувати. Зведений звіт тестування АСКП представлено на рисунку 3.5.

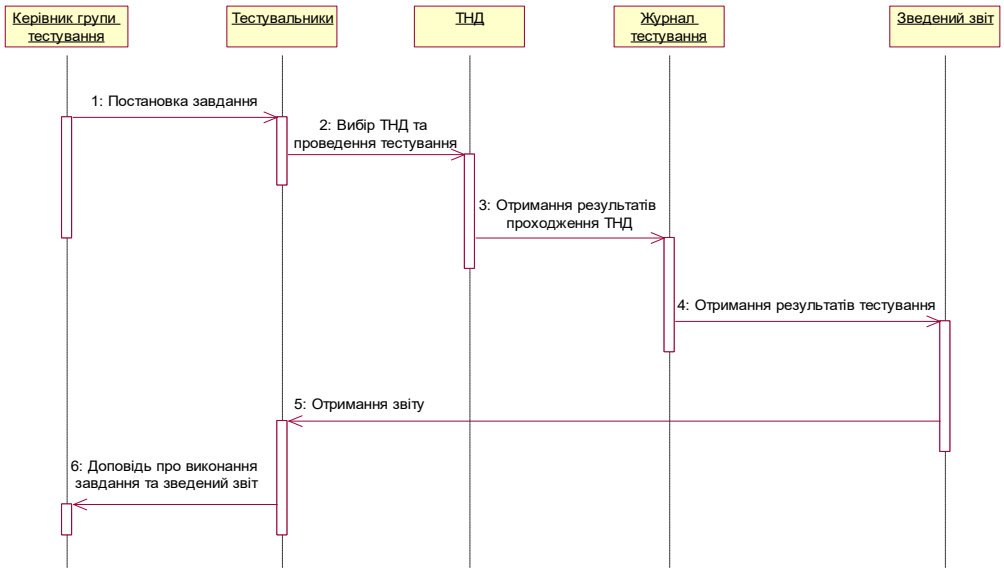

Рис 3.5. Зведений звіт по тестуванню АСКП

Таблиця 3.2. Застосування підходів/методів на рівнях тестування

Метод/підхід	Автономний	Інтеграційний	ПЗ	Системне
Методи структурного тестування	+	+		
Евристичні			+	+
Таблиці рішень	+	+		
Функціональні діаграми	+	+	+	
Еквівалентне розбиття	+	+	+	
Аналіз граничних значень	+	+	+	
Розбиття на категорії	+	+	+	
Тестування переходів станів	+	+	+	+
Тестування по специфікаціям	+	+	+	
Припущення про помилку	+	+	+	
Випадкове тестування			+	+
Операційний профіль				+
Тестування за сценаріями			+	+

Завдання 3.8. Проектування і розробка тестів.

Полягає в розробці проектів тестів, описів наборів тестів, процедур і сценаріїв тестування, відповідно до вибраних стратегій. У проекті тестів уточнюється підхід/метод тестування, визначається перелік функцій, тестованих цим методом, і вказуються набори тестів і процедури, по яким виконуватиметься тестування функцій і їх комбінацій. Проект тестів може бути розроблений для одного незалежного компонента ПЗ (наприклад, генератора звітів, інтерфейса користувача) або для групи зв'язаних тестів.

Структура цього документу, IEEE Std. 829, що рекомендується, приведена нижче.

Ідентифікатор проекту тестів

Унікальний ідентифікатор проекту тестів.

Тестовані характеристики

Вказують елементи і описують характеристики і комбінації характеристик, що охоплюється даним проектом. Для кожної характеристики (або комбінації) вказують посилання на пов'язані з нею вимоги в специфікації вимог або описі проекту.

Уточнений підхід

Перераховують використовувані методи тестування і аналізу результатів виконання тестів (наприклад, вживання утиліт порівняння файлів, візуальна перевірка), критерії підтвердження виконання тестів. Вказують загальні характеристики будь-яких наборів тестів. Наприклад, обмеження на допустимі входи для кожного набору тестів, умови середовища, додаткові процедурні вимоги і будь-які залежності тестів.

Ідентифікації тестів

Перераховують ідентифікатори і короткий опис кожного набору тестів, пов'язаного з даним проектом. Окремі набори тестів можуть використовуватися більш ніж в одному описі проекту. Перераховують ідентифікатори і короткий опис кожної процедури, зв'язаної з даним проектом.

Критерії проходження тестів

Вказують критерії, використовувані для визначення факту, чи пройшла конкретна характеристика або їх комбінація тести.

Опис набору тестів призначений для визначення конкретного набору тестів, пов'язаних з проектом тестів. Структура цього документа, що рекомендується стандартом IEEE Std. 829 приведена нижче.

Ідентифікатор тесту

Унікальний ідентифікатор набору тестів

Тестовані елементи

Перераховують і коротко описують елементи, що перевіряються, і характеристики. Для кожного елементу вказують заслання на відповідну документацію.

Опис входів. Вказують вагу входи/умови, потрібні для виконання набору тестів. Входи можуть вказуватися значеннями вхідних даних або іменами (для таблиць, файлів транзакцій). Вказують вагу використовувані бази даних, файли, повідомлення, області пам'яті. Вказують будь-які необхідні залежності між входами (наприклад, за часом).

Опис результатів. Вказують всі результати і характеристики (наприклад, час відповіді), потрібні для тестованого елементу. Для кожного результату або характеристики вказують точні значення (з допустимою погрішністю).

Умови середовища. Вказують вимоги до характеристик апаратної конфігурації. Вказують вимоги до системного і прикладного ПЗ необхідні для виконання наборів тестів (наприклад ОС, компілятори, інструменти тестування. СУБД і ін.). Будь-які інші спеціальні вимоги або вимоги до кваліфікації персоналу.

Особливі процедурні вимоги. Вказують будь-які обмеження, що стосуються процедур виконання наборів тестів (наприклад, налаштування, умови втручання оператора, правила зупинки тестів).

Залежності між тестами. Перераховують ідентифікатори наборів тестів, які повинні виконуватися перед даним набором тестів, і вказують причини.

Проста структура опису набору тестів може бути представлена у вигляді таблиці MS Excel (рисунок 3.6)

Дата: _____ Автор _____
Ідентифікатор тесту _____ Тип _____
Опис _____
ПС/Версія/ Модуль/Об'єкт _____
Мета тестування: _____
Пріоритет: _____
Передумова виконання: _____
Опис тестових умов: _____
а. Правильні умови _____
б. Не правильні умови _____
Тестові дані:
а. для правильних умов: _____
б. для не правильних умов: _____
Очікувана поведінка (результати):
а. для правильних умов: _____
б. для не правильних умов: _____
Отримані результати:
Відхилення:
Ідентифікатор проблеми: _____
Ідентифікатор тестової процедури: _____

Рис.3.6. Спрощена структура опису набору тестів

Стандартна структура більше підходить для формального тестування (при проведенні випробувань), для тестування критичних компонентів або тестування в рамках процесів V&V. У останніх випадках – можна використовувати для опису наборів тестів структуру, представлену на рисунку 3.6).

Процедура тестування - це документ, що містить детальні інструкції для виконання наборів тестів. Структура цього документа, що рекомендується стандартом IEEE Std. 829, приведена нижче, а на рисунку 3.7 представлена простіша структура опису процедури тестування.

Ідентифікатор процедури тестування

Унікальний ідентифікатор процедури.

Призначення

Вказують призначення процедури і посилання на пов'язані з нею набори тестів, а також посилання на відповідні розділи документації (наприклад, на процедури використання тестованого елементу).

Спеціальні вимоги

Вказують перелік всіх процедур, передуючих даній, вимоги до кваліфікації виконавців, вимоги до апаратно-програмного середовища і інструментів.

Кроки виконання процедури

Перераховують дії, необхідні для виконання процедури:

- реєстрація. Вказують методи і формати протоколювання результатів виконання тестів, фіксації проблем і будь-яких подій, що стосуються виконання процедури;
- встановлення/налаштування. Вказують дії, необхідні для підготовки до виконання процедури;
- запуск. Вказують дії, необхідні спершу виконання процедури;
- виконання. Вказують дії, необхідні під час виконання процедури;
- виміри. Вказують методи виконання вимірів результатів роботи тесту (наприклад, вказують, як вимірюється час виконання);
- призупинення. Вказують, як перервати виконання тесту;
- відновлення. Вказують будь-які передбачені точки відновлення роботи процедури і дії, потрібні для відновлення процедури в кожній точці;
- зупинка тесту. Вказують дії, потрібні для завершення роботи процедури;
- відновлення. Описують дії, необхідні для відновлення середовища;
- непередбачені події. Описують дії, що виконуються при виникненні аномальних подій при виконанні процедури.

Дата:_____ Автор: _____
Ідентифікатор процедури:_____ Тип _____
Опис: _____
Мета тестування (тестова вимога): _____ Пріоритет:_____
Файл специфікації _____
Ідентифікатори тестів:_____

Рис. 3.7. Спрощена структура процедури тестування

Окрім документів, що рекомендуються стандартами, для об'єднання логічно зв'язаних процедур можуть розроблятися сценарії тестування.

Пропонована структура опису сценарію тестування приведена нижче.

Ідентифікатор сценарію: _____

ПС/Версія: _____

Загальний опис:

Вказують, як використовувати сценарій

Передумови виконання:

Вказують дії з налаштування середовища і підготовці до виконання тестів (наприклад, мають бути заповнені файли або БД).

Постумови виконання:

Що необхідно робити після виконання сценарію (видалити тестові записи і ін.).

Виконання:

Вказують покрокові інструкції для виконання кожного набору тестів з відмітками про результати.

Час:

Вказують очікуваний час виконання сценарію.

Примітки:

Вказують будь-які розбіжності між отриманими і очікуваними результатами, а також описують будь-які проблеми, що виникли при виконанні тестів.

Завдання 3.9. Підготовка тестових даних.

Полягає в підготовці наборів даних (наприклад, у файлах, БД), необхідних для виконання тестів. Тестові дані мають бути максимально наближені до реальних і включати допустимі, граничні і недопустимі значення.

Завдання 3.10. Підготовка журналів по тестуванню.

Журнали призначені для реєстрації виконуваних тестів. Заповнення журналів здійснюється по ходу тестування.

Структура журналу тестування, що рекомендується стандартом IEEE Std. 829, представлена нижче. За бажання, можна використовувати структуру журналу, пропоновану стандартом ДСТУ 2851:

Ідентифікатор журналу. Унікальний ідентифікатор журналу.

Загальні відомості

Перераховують тестовані елементи з вказівкою їх версії і заслання на паспорт об'єкту. Вказують умови апаратно-програмного середовища тестування.

Реєстрація робіт і результатів

Вказують дату і час початку і завершення роботи (запуску і завершення тесту), а також прізвище оператора, реєструючого події.

Опис виконання

Вказують ідентифікатори виконуваних процедур тестування і посилання на їх опис. Перераховують учасників: тестувальників, операторів, спостерігачів і їх функції.

Результати виконання процедури

Для кожного тесту фіксують видимі результати (очікувані/не очікувані), а також місце їх розміщення (екран, файл.). Відзначають успішно або не успішно виконаний тест.

Середовище тестування

Вказують будь-які умови середовища, що стосуються виконання тесту.

Аномальні події

Для кожної події, поява якої не очікувалася, фіксують попередн. і наступну подію/умову. Фіксують будь-які обставини, що стосуються неможливості почати виконання процедури або її завершити.

Ідентифікатори звітів про проблеми

Перераховують ідентифікатори звітів про проблеми (інциденти), при їх виникненні.

Завдання 3.11. Перевірка тестових документів.

Мета перевірки - гарантувати, що всі цілі тестування (тестовані функції і області ризику відмови) охоплені наборами тестів. Це завдання може також виконуватися в рамках процесу верифікації.

Для перевірки може використовуватися таблиця покриття (табл. 3.3)

Таблиця 3.3. Покриття функцій наборами тестів

Функції		**Оцінка ризику**	**Ідентифікатор набору тестів**
Ідентифікатор	**Опис**		

При перевірці можна використовувати контрольні запитання:

♦ Чи охоплюють набори тестів вхідні дані з максимальними, мінімальними і нормальними значеннями?

♦ Чи включені в набір граничні і недопустимі данні/умови? Чи можна придумати інші неправильні вхідні умови, не охоплені тестами?

♦ Чи включені в набір, тести перевірки повідомлень про помилки?

♦ Чи включені тести для виявлення можливих помилок?

У інструментах управління тестуванням з кожною функцією, вказаною в ієрархії цілей, можна зв'язати набір тестів і процедури.

3.4. Автономне і інтеграційне тестування. Тестування програмного забезпечення системи

Завдання 4.1. Автономне тестування.

Виконується програмістом паралельно з розробкою і налагодженням програмних компонентів, з використанням основних методів тестування (функціонального і структурного). Методами функціонального тестування програміст повинен гарантувати, що кожна функція компонента протестована на допустимих, граничних і недопустимих даних. Методами структурного тестування, він повинен переконатися в тому. що всі оператори коду і умови протестовані не менше 1 разу, всі виклики і звернення до процедур, модулів і функцій компонента протестовані не менше 1 разу. Методом припущення про дефекти можуть бути розроблені додаткові тести для виявлення очікуваних помилок.

Детальні рекомендації по виконанню автономного тестування приведені в стандарті ANSI/IEEE Std. 1008.

Завдання 4.2. Повторне тестування після усунення дефектів.

Виконується для перевірки того, що виправлення зроблені правильно, і що нові помилки не внесені.

На рівні автономного тестування відмови і дефекти не фіксуються у формі звітів про проблеми, а результати виправлення наголошуються в журналі (у таблиці Excel).

Завдання 4.3. Аналіз результатів автономного тестування.

Для визначення схильних до дефектів модулів і уточнення оцінок ризику відмови можуть використовуватися метрики підрахунку дефектів і розподілу дефектів по модулях (профілі дефектів). Для таких модулів розробляються додаткові тести.

За результатами автономного тестування розробник готує звіт, в якому вказуються:

- досягнута повнота по метриках структурного і функціонального покриття;
- стан всіх протестованих модулів з вказівкою дефектів, що залишилися;

- недостатньо протестовані модулі і області (дані і умови середовища) і причини, по яких вони не були протестовані (брак часу, низький ризик відмови);
- можливі ризики відмови, пов'язані з дефектами, що залишилися;
- реальний час і трудомісткість тестування.

В разі передчасного завершення процесу через нестачу часу або виявлення дефектів, що вимагають змін в проекті, необхідно відобразити цю інформацію в звіті з вказівкою всіх не усунених дефектів.

Завдання 4.1 – 4.3 виконуються циклічно (окрім підготовки звіту) до досягнення встановленого критерію завершення тестування. Мінімальні цілі, які мають бути досягнуті при автономному тестуванні:

- 100% покриття операторів і не менше 85% умов;
- всі заплановані функціональні тести пройшли (всі області даних, схильні до відмов охоплені);
- немає не усунених серйозних дефектів;
- загальна стійка робота компонента в середовищі розробки.

Завдання 4.4. Інтеграційне тестування.

Інтеграція і тестування може виконуватися програмістами або групою тестування, залежно від умов інтеграції і «поширеності» проекту, але в будь-якому випадку інтегровані компоненти повинні знаходитися під контролем конфігурації.

Це означає, що всі зміни в спільно використовуваних бібліотеках і схемах БД, а також інтерфейсах між компонентами повинні виконуватися погоджено, а виявлені відмови фіксуватися в журналі тестування.

Завдання 4.5. Повторне тестування після усунення дефектів.

Завдання виконується циклічно до досягнення встановленого критерію завершення тестування (версія ПЗ зібрана). Повторюються лише тести інтеграції, при виконанні яких були виявлені проблеми.

Завдання 4.6. Аналіз результатів інтеграційного тестування.

Рішення задачі полягає в тому, аби визначити міру готовності інтегрованої версії ПЗ до функціонального тестування.

За результатами інтеграційного тестування група тестування готує звіт і паспорт об'єкту тестування (версії ПЗ).

Структура паспорта протестованого об'єкту, IEEE Std. 829, представлена нижче. Паспорт об'єкту підтверджує його готовність до тестування і служить супровідним документом цього об'єкту.

Ідентифікатор паспорта об'єкту

Унікальний ідентифікатор об'єкту.

Передавані елементи

Перераховують передавані на тестування елементи з вказівкою їх версії і посилань на відповідні документи по цих елементах і план тестування. Вказують відповідальних за передавані елементи.

Місцезнаходження елементів

Вказують носії, на яких розміщені елементи, імена передаваних файлів і бібліотек, дороги доступу.

Стан елементів

Вказують стан передаваних елементів, будь-які відхилення від проектної документації і плану тестування.

Підписи

Вказують підпис відповідального за передачу.

З моменту визначення версії ПЗ як базової, вона відсторонюється від розробників і «заморожується». Внесення змін до базової версії повинне виконується лише з метою виправлення дефектів і на підставі звітів про проблеми, виявлених тестувальниками. При тривалому тестуванні разом зі звітом про кожну проблему повинні складатися проміжні звіти про хід тестування. Після закінчення функціонального тестування готується звідний звіт про виконаний цикл функціонального тестування.

Завдання 4.7. Затвердження середовища тестування.

Перед початком тестування має бути створене відповідне апаратне і програмне середовище, що включає, наприклад, робочі станції, сервер, потрібні версії компонентів загальносистемного ПЗ і так далі, а також встановлені і конфігуровані інструменти тестування. Необхідно перевірити, що середовище встановлене і відповідає потрібному. Будь-які відхилення або затримки в установці повинні фіксуватися і розглядатися як можливий ризик зриву процесу (наприклад, деякі тести не будуть виконані). Можлива структура протоколу приведена нижче:

Протокол затвердження середовища тестування

Дата: _____
Шифр (ідентифікатор) проекту:___

Рівень та вид тестування	**Конфігурація середовища**	**Встановлена конфігурація**	**Можливі наслідки**

Затверджено:
_____ Дата:_____

Завдання 4.8. Затвердження ресурсів тестування.

Необхідно перевірити, що всі необхідні ресурси (трудові, фінансові, тимчасові) виділені відповідно до планів і достатні для проведення тестування. Будь-які відхилення від плану, наприклад, зрушення термінів передачі версії ПЗ групі тестування, брак тестувальників, повинні фіксуватися і розглядатися як можливий ризик зриву процесу. Можлива структура протоколу приведена нижче.

Протокол затвердження ресурсів тестування

Дата: _____
Шифр (ідентифікатор) проекту: ____

Рівень та вид тестування	**Необхідні ресурси**	**Фактичні ресурси**	**Можливі наслідки**

Затверджено:
_____ Дата: _____

Завдання 4.9. Тестування ПЗ.

Група тестування використовує проекти і описи наборів тестів, а також процедури, засновані на впорядкованих по критичності сценаріях функціонального тестування ПЗ.

Окрім функціонального тестування на цьому рівні виконується перевірка документації на ПЗ (посібники користувача) і довідкової системи (Help) по критеріях:

- простота використання і зрозумілість. Визначається, наскільки легко знайти потрібні відомості;достовірність. Виконуються вага описані в документації приклади і п
- роцедури користувача і звіряються ПЗ. документація і Help;
- повнота і точність. Визначається, чи вага функції і дії описані? Чи всі посилання вказані правильно?

Тестування по документації допомагає виявити дефекти ПЗ. Всі результати виконання тестів протоколюються в журналі тестування, а будь-які відхилення або відмови фіксуються в звітах про проблеми (рисунок 3.8).

Звіт про проблему

1. Ідентифікатор звіту: _____ 2. Ідентифікатор паспорта об'єкта_____

_____

3. Ідентифікатор тесту _____

4. Дата: _____ 5. Час: _____

6. Проект: _____

7. Програма/Модуль: _____ 8. Версія: _____

9. Середовище тестування: _____ 10. Апаратне середовище: _____

11. Рівень тестування: _____ 12. Цикл: _____

13. Симптом: _____

14. Серйозність (1-4): _____

15. Частота появи: _____

16. Статус (відкрито/закрито): _____

17. Опис: _____

18. Укладач звіту _____ 19. Переданий: _____

20. Коментар:

21. Рішення: _____

22. Пріоритет усунення(1-5): _____

23. Розглянуто: _____ Дата _____

24. Перевірено _____ Дата: _____

25. Вважати відкладеним (так/ні): _____

Рис. 3.8. Структура звіту про проблему

Звіт про проблему. Це основний документ, призначений для взаємодії між розробниками і групою тестування. У нім фіксуються будь-які непередбачувані або некоректні результати виконання тестів, виявлені групою тестування. Основною метою складання звіту є виправлення виявленого дефекту, тому звіт повинен складатися відразу після виявлення проблеми, містити чіткий опис проблеми і спосіб її відтворення.

Перші три поля призначено для того, щоб зв'язати звіт з тестовими документами, які потрібні для аналізу проблеми програмістом.

Поля 4 і 5 (дата і час) служать для подальшого відстежування усунення проблеми (при аналізі результатів ці дані використовуються для визначення віку дефектів, аналізу тенденцій і динаміки виявлення дефектів).

Наступні поля (6-10) використовуються для ідентифікації об'єкту і середовища тестування.

Поля 11-12 потрібні для аналізу процесу тестування.

У полі 13 вказується короткий опис прояву проблеми (симптому), що визначається тестувальником, а в полі 14 - серйозність проблеми (критична, серйозна і ін.). У полі 15 вказується порядковий номер появи даної проблеми (при повторному її виявленні). У полі 16 вказується стан проблеми (відкрита або закрита). У полі 17 вказується опис проблеми і спосіб її відтворення.

У полі 18 вказується прізвище тестувальника (або іншого укладача звіту, наприклад, користувача), а в 19 - прізвище програміста. У полі 20 програміст може записати, як усунена проблема, чому не усунена або відкладена, і з чим вона пов'язана.

Поле 21 використовується для опису стану проблеми, наприклад «Відхилити», «Розглядається», «Не може бути виправлена», «Виправлена» і заповнюється програмістом (на відміну від поля 16), яке заповнюється тестувальником. У полі 22 вказується код пріоритету усунення. У полі 23 вказується підпис програміста, відповідального за усунення проблеми, а в полі 24 - тестувальника, що перевірив усунення дефекту. Поле 25 заповнюється тестувальником, якщо він не згоден з пріоритетом усунення «Відкласти», або «Відхилити», встановленим програмістом або керівником проекту.

Для відстежування динаміки виконання тестування використовуються звідні данні про виявлені дефекти у вигляді проміжних звітів групи тестування (наприклад, щонеділі). В результаті аналізу цих звітів можуть бути уточнені первичні оцінки ризиків відмови функцій, визначені недостатньо протестовані області ПЗ і для них розроблені і виконані додаткові тести.

Нижче наведено два типи тижневих звітів (проміжних) про результати тестування, які складаються на підставі звітів про проблеми.

Звіт про динаміку виявлення дефектів

Дата: _____ Тип звіту: тижневий

Ідентифікатор звіту по проблемі	**Дата виявлення**	**Опис**	**Серйозність**

Всього виявлено дефектів:__

З них:

Критичних: _____

Серйозних: _____

Незначних: _____

Тривалість тестування: _____

Звіт про динаміку усунення дефектів

Дата: _____ Тип звіту: тижневий

Ідентифікатор звіту по проблемі	**Дата відкриття**	**Опис**	**Пріоритет усунення**	**Дата закриття**

Всього виявлено: ___

Всього усунено: ___

Відкрито:___

Перший із звітів – це основний звіт тестера про результати тижневої роботи. У нім вказуються всі виявлені за тиждень проблеми, впорядковані по серйозності.

Другий вид звіту відображає динаміку усунення дефектів і складається тестувальником за даними про «закриті» звіти про проблеми.

Завдання 4.10. Повторне тестування після усунення дефектів.

Завдання виконується циклічно до досягнення заданого критерію завершення.

Завдання 4.11. Аналіз результатів тестування.

Після виконання запланованих тестів виконується аналіз покриття функцій набором тестів і аналіз не усунених дефектів, що залишилися. Визначається ризику відмов із-за дефектів, що залишилися, і приймається рішення про продовження або завершення тестування ПЗ. Після закінчення тестування група тестування готує звіт про виконання циклу тестування.

Звіт про виконання тестування ПЗ

Дата складання звіту: _____

Проект/Програма: __ Версія:_____ Цикл:_____

Дата початку тестування:__ Рівень/Задача тестування: _____

Серйозність дефектів	**Усунено**	**Відкладено**	**Відключено**
Критичні			
Серйозні			
Середні			
Інші			
Загалом			

Тривалість тестування (днів): _____

Трудомісткість тестування: __

Звіт склав: _____

Завдання 4.12. Проектування комплекту регресійних тестів для регресивного тестування. Мета: визначити набори тестів, сценарії і процедури, які мають бути збережені для регресійного тестування в наступній версії ПЗ.

Рішення має бути засноване на функціональних вимогах до системи, результатах аналізу ризиків і аналізі дефектів, які мали місце при функціональному тестуванні.

Завдання 4.13. Автоматизація наборів тестів для регресійного тестування (необов'язкова). При автоматизованому тестуванні процедури створюються у вигляді тестових скриптів (автоматизованих процедур). Ці скрипти можуть згенерувати автоматично шляхом «програвання» сценарію роботи або написані вручну на мові скриптів інструменту тестування. Тестовий скрипт повинен зберігатися і оновлюватися при кожному новому виконанні повторного тестування.

Зв'язані логікою роботи окремі процедури об'єднуються в тестові сценарії за допомогою створення процедур-оболонок (shell procedure).

ВИСНОВКИ

Оскільки програмні системи критичного призначення пов’язані з безпекою життєдіяльності, то проблема перевірки (верифікації) критичних ПС, а також їх тестування є актуальною. При атестаційних та сертифікаційних випробуваннях критичних програмних систем, які проводяться для забезпечення необхідного рівня їх якості, необхідно проводити всеосяжне тестування на області визначення вхідних даних домену (предметної області ПС).

Тестування проводиться в процесі випробувань ПС для визначення показників якості. Проблема, яка виникає при випробуваннях ПС, полягає в тому, що відомі стратегії та процедури тестування в основному орієнтовані на повне покриття елементів класів еквівалентності за вхідними даними і є трудомісткими і витратними. Крім того, використовувані методи часто дублюють один одного, а тестові набори даних перетинаються на множинах областей визначення реалізованих функцій ПС. Без сумніву, застосовані методи тестування мають охоплювати відомі класи еквівалентності досліджуваного домену, але таке покриття не повинно бути надмірним. В додаток до цього необхідно досягти високої здатності набору сценаріїв тестування (стратегії тестування) до виявлення ще не знайдених помилок у ПС, тобто високої детектабільності методології тестування.

У даній роботі в якості прикладу був розглянутий клас програмних систем контролю польотів літальних апаратів, які є критичними, оскільки пов'язані з безпекою життєдіяльності. Ці системи призначені для контролю діяльності екіпажів, двигунів і радіоелектронних компплексів ЛА. Однією з найбільш важливих проблем в процесі випробувань є організація тестування ПЗ контролю польотів, котре слугує для визначення фактичних значень показників якості ПЗ, що входять у модель. Результати недбалого і неповного тестування ПЗ систем контролю можуть мати серйозні наслідки, впритул до катастрофічних.

З вищевикладеного випливає, що поєднуючи методи створення тестів в складений метод (наприклад, еквівалентне розбиття + граничні значення, або комбінаторне покриття умов + динамічне стохастичне тестування) і переслідуючи мету створення безперервної повної вхідної області, ми натомість отримуємо збільшення міри детектабільності за рахунок збільшення кількості підобластей і тестових елементів в них. Показано, що включення в стратегію тестування критичних ПС великої кількості методів нераціональне і веде до зайвих матеріальних витрат, суттєво не збільшуючи загальну детектабільноіть стратегії. У роботі на основі аналізу галузі застосування ПЗ АСКП обгрунтовано вибір з існуючих методів тестування тих методів, які при відсутності зайвого дублювання даних в ТНД забезпечують виявлення найбільшої кількості невідповідностей ПЗ специфікаціям і дефектів у модулях ПЗ АСКП. При оцінці побудованої стратегії тестування ПЗ АСКП отримані високі показники ймовірності виявлення помилок і невідповідностей ПЗ вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСТУ 2462–94. Сертифікація. Основні поняття. Терміни та визначення. – Чинний від 01.01.95. — К. : Держстандарт України, 1994. — 27 с.

2. ДСТУ 2853–94. Програмні засоби ЕОМ. Підготовлення і проведення випробувань. — Чинний від 01.01.96. — К .: Держстандарт України, 1994. — 17 с.

3. ДСТУ 2851–94. Програмні засоби ЕОМ. Документування результатів випробувань. — Чинний від 01.01.96. — К. : Держстандарт України, 1994. — 12 с.

4. ISO/IEC 12119. IT — Software packages — Quality requirements and testing, 1994. — 29 p.

5. ISO/IEC 9126-1. Software engineering — Product quality — Part 1: Quality model, 2001. — 26 p.

6. Основы инженерии качества программных систем / Ф.И. Андон, Г.И. Коваль., Т.М. Коро- тун, Е.М. Лаврищева, В.Ю. Суслов. — Киев : Академпериодика, 2007. — 672 с.

7. Брауде Э. Технология разработки программного обеспечения. — СПб. : Питер, 2004. — 655 с.

8. Канер Сэм, Фолк Дж., Нгуен Енг Кен. Тестирование программного обеспечения : Пер. с англ. — Киев : Диасофт, 2001. — 544 с.

9. Майерс Г. Искусство тестирования программ : Пер. с англ. — М. : Финансы и статистика, 1982. — 176 с.

10. Дастин Э., Рэшка Д., Пол Дж. Автоматизированное тестирование программного обеспечения. Внедрение, управление и эксплуатация : Пер. с англ. — М. : Лори, 2003. — 567с.

11. Мороз Г.Б. Надежность и трастабильность программных средств высокоцелостных систем // УСиМ. — 1999. — № 2. — С. 59–68.

12. Мороз Г.Б. Некоторые случаи оптимизации трастабильности программных средств // Проблемы программирования. — 2000. — № 1–2. — С. 361–366.

13. ДСТУ 3275–95. Системи автоматизованого оброблення польотної інформації наземні. Загальні вимоги. — Чинний від 01.07.96. – К. Держстандарт України, 1996. — 7 с.

14. Замковий В.В., Харченко О.Г., Галай І.О. Використання моделей якості для специфікації вимог при проектування програмних систем // Зб. наукових праць Камя’нець-Подільського нац. ун-ту. — 2008. — № 11. — С. 131–137.

15. Райчев І.Е. Проблеми сертифікації програмного забезпечення автоматизованих систем контролю // Вісник НАУ. — 2004. — № 1. — С. 23–28.

16. Райчев І.Е., Харченко О.Г. Концепція побудови сертифікаційної моделі якості програмних систем // Проблемы программирования. — 2006. — № 2–3. — С. 275–281.

17. IEC 6150 : Functional Safety of electrical / electronic / progammable electronic safety-related systems.

18. Соммервилл И. Инженерия программного обеспечения : Пер. с англ. – М. Вильямс, 2002.— 624 с.

19. Райчев И.Э., Харченко А.Г., Яцков Н.А. Исследование методов тестирования программных модулей обработки полетной информации // Вісник КМУЦА. – 2000. – № 1–2. — С. 127–133.

20. Райчев И.Э., Харченко А.Г., Яцков Н.А. Методы создания тестовых наборов данных при сертификационных испытаниях комплексов программ контроля полетов // Вісник НАУ. — 2001. — № 1. — С. 126–132.