

Міністерство освіти і науки України
Запорізький національний технічний університет



Co-funded by the
Tempus Programme
of the European Union

Г.В. Табунщик, Р.К. Кудерметов, Т.І. Каплієнко

Інженерія якості програмного забезпечення

Навчальний посібник

Рекомендовано Міністерством освіти і науки України як навчальний посібник для студентів вищих навчальних закладів, які навчаються за освітньо-професійною програмою магістра за спеціальністю «Програмне забезпечення систем»

УДК 004.4 (075.8)
ББК 32.972–02 я73
Т-12

Рекомендовано Міністерством освіти і науки України як навчальний посібник для студентів ВНЗ, які навчаються за освітньо-професійною програмою магістра за спеціальністю «Програмне забезпечення систем» (лист №1/11–6818 від 03.03.2013 р.)

Р е ц е н з е н т и :

ГОМЕНЮКС. І., доктор технічних наук, професор, декан математичного факультету Запорізького національного університету.

КУПН А.І., доктор технічних наук, доцент, проф. кафедри комп'ютерних систем та мереж, декан факультету інформаційних технологій ДВНЗ «Криворізький технічний університет».

СИТНИКОВ В.С., доктор технічних наук, професор, зав. кафедри комп'ютерних систем ОНПУ.

Табунщик Г.В., Кудерметов Р.К., Каплієнко Т.І.

Т-12 Інженерія якості програмного забезпечення: навчальний посібник, 2-ге видання. – Запоріжжя: Дике Поле, 2016. – 176 с.

ISBN 978-966-2752-94-6

Книга містить основні концепції керування якістю, зокрема якістю програмного забезпечення, на протязі всього життєвого циклу. Розглянуті питання сертифікації та оцінки витрат на якість. Особливу увагу приділено метрикам якості інформаційних систем, та їх використанню. Видання призначене для студентів комп'ютерних спеціальностей вищих навчальних закладів, а також може бути використаним аспірантами, науковими та педагогічними працівниками, фахівцями-практиками.

УДК 004.41 (075.8)
ББК 32.972–02 я73

Посібник видано за підтримки міжнародного проекту ICo-op «Промислове співробітництво та креативна інженерна освіта на основі дистанційного інженерного та віртуального інструментарію» (530278-TEMPUS-1-2012-1-DETEMPUS-JPHES) за програмою TEMPUS Європейської комісії.

Проект фінансується за підтримки Європейської Комісії. Зміст даної публікації/матеріалу відображає думку авторів і Європейська Комісія не несе відповідальність за використання інформації, що в ній міститься.

ISBN 978-966-2752-94-6

© Запорізький національний
технічний університет (ЗНТУ), 2016

ЗМІСТ

Вступ.....	6
Частина 1 Основні концепції забезпечення якості інформаційних систем	
1. Життєвий цикл інформаційних систем.....	8
1.1 Етапи життєвого циклу програмного забезпечення.....	8
1.2. Моделі програмних проєктів з ітеративним життєвим циклом .	14
1.2.1 Методологія Rational Unified Process.....	14
1.2.2 Модель Microsoft Solutions Framework.....	17
1.2.3 Модель Personal Software Process/Team Software Process.....	18
1.2.4 Гнучка методологія розроблення програмного забезпечення (Agile).....	19
1.2.4.1 Екстремальне програмування eXtreme Programming	20
1.2.4.2 Сімейство методологій Crystal	22
1.2.4.3 Методологія Feature Driven Development	23
1.2.4.4 Методологія Scrum	23
1.2.4.5 Методика Kanban.....	31
1.3 Практичні завдання	35
1.4 Контрольні запитання	35
1.5 Література до розділу.....	36
2 Організація забезпечення якості та сертифікація програмних продуктів	37
2.1 Об'єкт, процес та система керування якістю програмних систем	37
2.2 Стандарти для систем керування якістю ПЗ.....	40
2.2.1 Міжнародний стандарт ISO 9001	40
2.2.2 Capability Maturity Model / Capability Maturity Model Integration	43
2.2.3 Стандарт версії ISO/IEC 15504	49
2.3 Сертифікація програмних продуктів	53
2.3.1 Процес сертифікації програмних продуктів	53
2.3.2 Документування процесів і результатів сертифікації програмних продуктів	60
2.4 Практичні завдання	64
2.5 Контрольні запитання	64
2.6 Література до розділу.....	65
3 Витрати на якість.....	66
3.1 Вартість якості	66

3.2 Фунції втрат	68
3.3 Технології оцінювання трудовитрат	70
3.4 Практичні завдання	72
3.5 Контрольні запитання	73
3.6 Література до розділу	73
Частина 2 Метрики якості інформаційних систем	
4 Моделі якості програмного забезпечення	74
4.1 Модель якості за МакКолом	74
4.2 Модель якості за Боемом	76
4.3 Модель якості FURPS	76
4.4 Модель якості ISO 9126	77
4.5 Практичні завдання	81
4.6 Контрольні запитання	81
4.7 Література до розділу	81
5 Метрика як основа вимірювання	82
5.1 Метрика як основа виміру	82
5.1.2. Класифікація мір якості	84
5.1.3 Класифікація метрик якості	85
5.2 Проектування метрик якості	89
5.4 Практичні завдання	92
5.5 Контрольні запитання	93
5.6 Література до розділу	93
6 Метрики коду програмного забезпечення	94
6.1 Кількісні метрики	94
6.3 Метрики складності потоку керування програми	98
6.4 Метрики складності потоку керування даними	103
6.5 Метрики складності потоку керування і даних програми	106
6.6 Об'єктно-орієнтовані метрики	110
6.7 Гібридні метрики	114
6.8 Практичні завдання	115
6.9 Контрольні запитання	115
6.10 Література до розділу	116
7 Метрики якості програмних проєктів	117
7.1 Оцінювання якості показників керування	118
7.1.1 Оцінювання якості роботи та прогресу	118
7.1.2 Оцінювання фінансового стану проєкту	120
7.1.3 Оцінювання найму персоналу	122
7.2 Оцінювання показників якості розробки	124

7.2.1 Оцінювання інтенсивності змін і стабільності	124
7.2.2 Дефекти та коефіцієнт дефективності	125
7.2.3 Оцінювання вартості доробки і адаптованості	126
7.2.4 Оцінювання середнього часу до відмови	127
7.3 Використання метрик ПЗ.....	130
7.4 Практичні завдання	132
7.5 Контрольні запитання	133
7.6 Література до розділу.....	133
8 Керування конфігурацією програмних засобів	134
8.1 Основні визначення.....	134
8.2 Функціонування систем керування конфігурацією.....	137
8.3 Практичні завдання	140
8.4 Контрольні запитання	140
8.3 Література до розділу.....	140
Перелік посилань.....	141
Алфавітно-предметний покажчик.....	155
Додаток А Історія розвитку систем керування якістю	158
Додаток Б Моделі оцінки вартості.....	160
Б.1 Конструктивна модель вартості	161
Б.2 Методологія функціональних точок для оцінювання вартості розроблення програмного забезпечення	170
Додаток В Перелік тем для практичних завдань	180

ВСТУП

Діяльність розробників програмних засобів регламентується низкою методів та стандартів, які є компонентами технологічного забезпечення інформаційних систем. Тому для студентів комп'ютерних спеціальностей важливо володіти знаннями про стандарти, які дозволяють забезпечити якість на всіх етапах життєвого циклу програмних засобів.

Серед метрик якості програмного забезпечення є універсальні метрики, які можливо застосовувати практично до всіх видів програмного забезпечення. У той же час велика частина найбільш важливих метрик в успішних проектах розробляється індивідуально на основі особливостей проекту і характеристик предметної області. Тому важливим для студентів з програмної інженерії та комп'ютерних наук є вивчення досвіду з використанням метрик якості.

Мета книги – систематизувати знання в області керування якістю, зокрема якістю програмних продуктів.

Викладений матеріал був апробований під час проведення дисциплін «Якість інформаційних систем», «Моделі та метрики програмних систем», що викладались студентам спеціальностей «Програмне забезпечення систем» та «Інформаційні технології проектування».

Посібник було розроблено під час виконання Міжнародного проекту «Practice oriented Master Programmes in Engineering in RU, UA, UZ (PROMENG)» (JPCR – 510920) за програмою «ТЕМПУС» Європейської Комісії.

Структурно підручник містить дві основні частини. Перша частина присвячена основним поняттям якості програмних засобів.

Перший розділ присвячений життєвому циклу програмних засобів та його інтерпретаціям в залежності від моделі керування, що використовується. Другий розділ містить загальні відомості з керування та забезпечення якістю програмних засобів, стандарти на системи якості та порядок сертифікації програмних систем. третій розділ містить визначення ціни якості та функцій втрат.

Друга частина посібника присвячена метрикам якості інформаційних систем.

У четвертому розділі наведено еволюцію моделей якості інформаційних систем.

Розділи п'ять, шість та сім присвячені огляду різних класів метрик, їх детальному опису, розглянута роль метрик при розробленні програмного забезпечення.

Розділ вісім містить основні поняття з керування конфігурацією та модель функціонування системи керування конфігурацією.

Видання орієнтоване на студентів комп'ютерних спеціальностей вищих навчальних закладів, а також може використовуватися аспірантами, науковими та педагогічними працівниками, практичними фахівцями.

Терміни, визначення яких наводяться в тексті виділено жирним курсивом. Кожний розділ закінчується контрольними запитаннями та практичними завданнями, що можуть бути використані як для самоперевірки, так і при підготовці лабораторних, практичних та контрольних завдань.

ЧАСТИНА 1 ОСНОВНІ КОНЦЕПЦІЇ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ІНФОРМАЦІЙНИХ СИСТЕМ

1. ЖИТТЄВИЙ ЦИКЛ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1 Етапи життєвого циклу програмного забезпечення

Розроблення програмного забезпечення відповідає певному життєвому циклу. **Життєвий цикл (ЖЦ)** – це впорядкований набір видів діяльності, здійснюваний та керований у рамках кожного проекту з розроблення програмного забезпечення (ПЗ).

На узагальненому рівні життєвий цикл може містити лише три етапи:

- аналіз;
- проектування;
- реалізація.

Етап аналізу концентрується на системних вимогах. Вимоги визначаються та специфікуються. Здійснюється розроблення та інтеграція функціональних моделей та моделей даних для системи. Крім того, фіксуються не функціональні вимоги та інші системні обмеження.

Етап проектування поділяється на архітектурне та деталізоване проектування. Зокрема, проводиться уточнення конструкції програми для архітектури клієнт/сервер, що інтегрує об'єкти інтерфейсу користувача та бази даних. Ставляться та фіксуються питання проектування, що впливають на зрозумілість, пристосованість до супроводу та масштабованість системи.

Етап реалізації містить створення коду клієнтських програмних додатків і серверів баз даних. Коротко кажучи, аналіз вказує на те, що робити; проектування – на те, як за допомогою наявної технології зробити це, а реалізація втілює задумане на попередніх етапах.

Основна мета сучасних технологій підтримки ЖЦ ПЗ полягає в забезпеченні економічної, технічної та соціальної ефективності всього життєвого циклу інформаційних систем у різних проблемно-

орієнтованих галузях. До поняття сучасної технології відноситься сукупність методів і інструментальних засобів автоматизації, а також технологічні процеси, що забезпечують життєвий цикл складних ПЗ із заданими функціональними і конструктивними характеристиками якості.

Діяльність розробників ПЗ регламентується низкою методів і стандартів, які є компонентами технологічного забезпечення складних ПЗ протягом їх життєвого циклу.

Методичною основою технології ЖЦ ПЗ, що регламентує діяльність фахівців, є *модель організації процесу розроблення*. Вона відображається сукупністю етапів і операцій у послідовності їх виконання та взаємозв'язку, які забезпечують впорядковане ведення робіт на всіх стадіях від ініціювання проекту і підготовки технічного завдання до завершення випробувань або застосування версії програмного продукту (ПП). Індустриалізація технологій створення ПЗ базується на стандартизації процесів розроблення програм, їх структурної побудови та інтерфейсів з операційним та зовнішнім середовищем. Для цього використовуються методи та підходи керування програмними проектами.

Методологія забезпечення якості ПЗ підтримана рядом методичних документів та інструментальних засобів, а також формалізовано групою міжнародних стандартів. Концептуальні та організаційні основи адміністративного керування життєвим циклом і якістю ПЗ визначені у восьми базових принципах, які декларовані в стандартах ISO 9000:2000 та ISO 15504-2-2009 і складають основу технологічних процесів у цих стандартах.

Принцип 1 – орієнтація підприємства-розробника на споживача-замовника. Підприємства залежать від своїх споживачів і, отже, повинні розуміти поточні та майбутні потреби споживачів-замовників, задовольняти їхні вимоги і прагнути перевершити їх очікування.

Принцип 2 – лідерство-інструкція. Лідери забезпечують єдність призначення та напрямки діяльності підприємства. Вони повинні створювати і підтримувати внутрішнє навколишнє середовище, в якому фахівці можуть повною мірою брати участь у досягненні стратегічних цілей підприємства.

Принцип 3 – залучення персоналу. Люди складають основу підприємства на всіх рівнях, і їх повноцінна участь у діяльності

компанії сприяє застосуванню їх здібностей на благо цілей підприємства.

Принцип 4 – підхід, заснований на процесах. Бажаний результат досягається більш ефективно, коли необхідні ресурси і діяльність фахівців підприємства керуються як єдиний зв'язаний процес.

Принцип 5 – системний підхід до адміністративного керування. Виявлення та розуміння завдань і адміністративне керування системою взаємопов'язаних процесів для заданої стратегічної мети підвищує ефективність і результативність підприємства.

Принцип 6 – постійне вдосконалення. Безперервне удосконалення процесів і підвищення якості продукції повинно бути постійною метою підприємства та його фахівців.

Принцип 7 – підхід до прийняття рішень, заснований на фактах. Ефективні рішення мають базуватися на аналізі лише реальних даних і достовірної інформації.

Принцип 8 – взаємовигідні відносини з постачальниками. Підприємство-користувач і його постачальники-розробники взаємозалежні, взаємовигідні відносини між ними підвищують здатність обох виробляти якісну продукцію.

Дотримання цих принципів сприяє підвищенню управлінської культури, застосуванню системи адміністративного керування якістю в усіх видах діяльності підприємств і, як наслідок, забезпеченню конкурентоспроможності створюваної продукції, проектів і систем.

Основними **цільми** впорядкування, регламентування процесів та застосування стандартів у **ЖЦ ПЗ** є:

- зниження трудомісткості, тривалості процесів, вартості та покращення інших техніко-економічних показників проектів програмних продуктів;
- підвищення якості розроблюваних і/або застосовуваних компонентів і програмних засобів в цілому при їх придбанні, розробці, супроводі та експлуатації;
- забезпечення можливості розширювати ПЗ з використання прикладних функцій і масштабувати комплекс програм залежно від зміни розмірності вирішуваних завдань;

– забезпечення переносимості прикладних програм і даних між різними апаратними та операційними платформами і повторного використання програмних компонентів.

Застосування стандартів ЖЦ дозволяє орієнтуватися на побудову систем і комплексів програм з великих функціональних вузлів, які відповідають вимогам стандартів, застосовувати відпрацьовані і перевірені проектні рішення. Вони визначають уніфіковані інтерфейси взаємодії компонентів таким чином, що розробнику системи не потрібно вдаватися в деталі їх внутрішнього устрою. Стандарти, пов'язані з програмним комплексом (функціональним частинам) систем, полегшують повторне використання в нових системах готових і апробованих програмних продуктів. Для уніфікації та регламентування процесів ЖЦ ПЗ такі сукупності (профілі) стандартів повинні адаптуватися і конкретизуватися стосовно до певних класів проектів, процесів і компонентів ПЗ. Розробка програмного продукту, в значній мірі, може зводитися до їх інтеграції та комплексування з стандартизованих компонентів.

Методи і процеси регламентування ЖЦ ПЗ відіграють стабілізуючу і організуючу роль у всьому ЖЦ багатьох складних систем. Вони забезпечують:

- розширення і вдосконалення функцій систем і компонентів із збереженням їх цілісності і первинних витрат;
- систематичне підвищення якості функціонування комплексів програм і баз даних, рішення задач користувачів в різній зовнішньому середовищі;
- поліпшення техніко-економічних характеристик застосування систем і програмних продуктів;
- вдосконалення технологій забезпечення життєвого циклу складних систем і комплексів програм.

Для цього при створенні і супроводі складних, розподілених систем, формуванні їх архітектури, виборі програмних компонентів і їх зв'язків доцільно враховувати **ряд сучасних концептуальних вимог формування їх ЖЦ:**

- архітектура комплексу програм повинна відповідати поточним і перспективним цілям і стратегічним функціональним завданням створюваної системи, бути досить гнучкою і допускати відносно просте, без докорінних структурних змін, розвиток та

нарощування функцій і ресурсів системи відповідно з розширенням сфер та завдань її застосування;

- у структурі та компонентах ПЗ і системи слід передбачати забезпечення максимально можливого зберігання інвестицій в апаратні і програмні засоби, а також в бази даних при тривалому розвитку, супроводження та модернізації системи;

- необхідно забезпечувати ефективне використання ресурсів в ЖЦ системи та мінімізувати інтегральні витрати на обробку даних в типових режимах її функціонування з урахуванням експлуатаційних витрат і капітальних вкладень у створення системи та програмного продукту;

- повинні бути забезпечені безпека функціонування системи і надійний захист даних від помилок, від руйнування або втрати інформації, а також авторизація користувачів, керування робочим завантаженням, резервуванням і оперативним відновленням функціонування системи та програмного продукту;

- для забезпечення перспективи розвитку життєвого циклу системи і комплексу програм доцільно передбачати можливість інтеграції гетерогенних обчислювальних компонентів і можливість перенесення ПЗ і баз даних (БД) на різні апаратні та операційні платформи на основі концепції та стандартів відкритих систем;

- слід забезпечити комфортне навчання і максимально спрощений доступ кінцевих користувачів до керування і результатів функціонування системи та програмного продукту на основі сучасних графічних засобів та наочних інтерфейсів користувача.

Високі темпи зростання основних ресурсів апаратних засобів і постійна потреба в збільшенні їх використання призводять до необхідності адекватного вдосконалення технологій створення програмних засобів та баз даних, що розвиваються. Гнучкість модифікації ПЗ в життєвому циклі забезпечується низкою принципів і правил структурної побудови та процесів життєвого циклу комплексів програм і їх компонентів, а також взаємодією між ними. Ці правила спрямовані на стандартизацію та уніфікацію структури і взаємодії компонентів різного рангу і призначення в межах проблемної області. Їх формалізація і виконання забезпечують значний ефект у зниженні трудомісткості і тривалості розроблення та модифікації ПЗ, БД та їх версій. Одним з найважливіших і ефективних шляхів вирішення цієї

проблеми є застосування концепції і сукупності стандартів відкритих систем.

В процесі експлуатації ПЗ у кожного користувача можуть з'являтися деякі претензії до функціонування, які кваліфікуються їм як помилки чи дефекти базової або власної адаптованої версії. Від користувачів або замовників можуть надходити також пропозиції по додатковому внесенню змін в базову версію для поліпшення експлуатаційних характеристик і розширення функціональних можливостей ПЗ. Аналогічні пропозиції можуть надходити від розробників комплексу програм. Для вирішення таких завдань розроблені і активно застосовуються в життєвому циклі стандартизовані методи, методики і засоби автоматизації регламентованого супроводу та керування конфігурацією. Вони дозволяють надати інформацію окремим фахівцям і керівникам про стан проекту і його компонентів в будь-який момент часу і не допускати хаосу при колективної модифікації програм і даних. Дисципліна в супроводі та конфігураційному управлінні в значній мірі визначає техніко-економічні показники життєвого циклу складного проекту, його якість, тривалість застосування і конкурентоспроможність програмного продукту.

Зростання складності та відповідальності сучасних завдань, що вирішуються великомасштабними системами, а також можливого збитку від недостатньої якості комплексів програм, значно підвищило актуальність освоєння методів стандартизованого опису вимог та оцінювання характеристик якості на різних етапах життєвого циклу ПЗ. Це стимулювало розвиток і застосування методів, стандартів і засобів автоматизації індустріальної програмної інженерії. Виявилася необхідність систематизації реальних характеристик якості ПЗ, застосування стандартів для вибору з них необхідної номенклатури і необхідних значень для конкретних проектів комплексів програм. Обіцянки розробників в контрактах із замовниками створити високоякісні ПЗ в узгоджені терміни у багатьох випадках не виконуються, як наслідок відмінностей у розумінні необхідної якості, так і внаслідок невміння оцінити ресурси, необхідні для досягнення заданого замовником якості програм. Стратегічною проблемою в життєвому циклі сучасних систем стало забезпечення необхідної якості великомасштабних ПЗ при реальних обмеженнях на використання ресурсів.

1.2. Моделі програмних проєктів з ітеративним життєвим циклом

Найбільш поширені об'єктно-орієнтовані моделі, орієнтовані на ітеративний процес розроблення – Rational Unified Process (RUP), Microsoft Solutions Framework (MSF), Personal Software Process / Team Software Process (PSP/TSP) і Agile (eXtreme Programming (XP), Crystal, Feature Driven Development (FDD), Scrum).

1.2.1 Методологія Rational Unified Process

Уніфікований процес *Rational Unified Process (RUP)* був розроблений Філіпом Крачтеном (*Philippe Kruchten*), Іваром Якобсоном (*Ivar Jacobson*) та іншими співробітниками компанії «Rational Software» як доповнення до мови моделювання *Unified Modeling Language (UML)*. Особливістю RUP є те, що в результаті роботи над проєктом створюються та вдосконалюються моделі ПЗ. Замість створення величезної кількості паперових документів, RUP спирається на розробку і розвиток семантично збагачених моделей, які всебічно представляють систему, що розробляється.

RUP, на відміну від більшості інших методологій, дозволяє в широкому діапазоні обирати ступінь формалізації і ітеративний процес розроблення в залежності від особливостей проєктів і організації. *RUP* найбільш часто використовують компанії з великою кількістю розробників (понад 40-50 чоловік).

ЖЦ проєкту *RUP* складається фаз і дисциплін. Фази *RUP*: початок (*Inception*), проектування (*Elaboration*), розробка (*Construction*) та впровадження (*Transition*). Послідовність цих фаз фіксована, але число ітерацій, необхідних для завершення кожної фази, визначається індивідуально для кожного конкретного проєкту. Кожна фаза складається з дисциплін: «Бізнес-моделювання» (*Business modeling*); «Керування вимогами» (*Requirements*); «Аналіз і Проектування» (*Analysis and Design*); «Реалізація» (*Implementation*); «Тестування» (*Test*); «Розгортання» (*Deployment*); «Керування проєктом» (*Project management*); «Керування змінами» (*Change management*); «Середовище» (*Environment*).

У *RUP* всі дисципліни виконуються практично у всіх фазах ЖЦПЗ. Однак, в залежності від фази, змінюються поточні цілі проекту і співвідношення між обсягами робіт, що відповідають різним матеріалам.

Розглянемо рольову структуру проекту, що запропонована Центром об'єктно-орієнтованої технології компанії IBM:

- замовник (*Customer*) реально існуючий (в організації, якій підпорядкована команда або поза нею) ініціатор розроблення або хто-небудь інший, уповноважений приймати результати (як поточні, так і остаточні) розроблення;

- планувальник ресурсів (*Planner*) висуває та координує вимоги до проектів в організації, що здійснює цю розробку, а також розвиває та спрямовує план виконання проекту з точки зору організації;

- менеджер проекту (*Project Manager*) відповідає за розвиток проекту в цілому, гарантує, що розподіл завдань і ресурсів дозволяє виконати проект, що роботи та подання результатів йдуть за графіком, що результати відповідають вимогам. У рамках цих функцій менеджер проекту взаємодіє з замовником і планувальником ресурсів;

- керівник команди (*Team Leader*) виконує технічне керівництво командою в процесі виконання проекту. Для великих проектів можливе залучення декількох керівників підкоманд, які відповідають за вирішення своїх завдань.

- архітектор (*Architect*) відповідає за проектування архітектури системи, погоджує розвиток робіт, пов'язаних з проектом.

- проектувальник підсистеми (*Designer*) відповідає за проектування підсистеми або категорії класів, визначає реалізацію та інтерфейси з іншими підсистемами.

- експерт з предметної області (*Domain Expert*) відповідає за вивчення сфери застосування, підтримує спрямованість проекту на вирішення завдань цієї галузі;

- розробник (*Developer*) реалізує спроектовані компоненти, має та створює специфічні класи та методи, здійснює кодування та автономне тестування, будує продукт. Це широке поняття, що може підрозділятися на спеціальні ролі (наприклад, розробник класів). Залежно від складності проекту, команда може включати різну кількість розробників;

- розробник інформаційної підтримки (*Information Developer*) створює документацію, що супроводжує продукт, коли випускається

версія. До неї включаються інсталяційні, інформаційні та навчальні матеріали. Матеріали допомоги надаються на паперових і машинних носіях. Для складних проектів можливий розподіл цих завдань між кількома розробниками інформаційної підтримки;

- спеціаліст з інтерфейсу користувача (*Human Factors Engineer*) відповідає за зручність застосування системи. Працює з замовником, щоб упевнитися, що призначений для користувача інтерфейс задовольняє вимогам;

- тестувальник (*Tester*) перевіряє функціональність, якість та ефективність продукту; будує та виконує тести для кожної фази розвитку проекту;

- бібліотекар (*Librarian*) відповідає за створення та ведення спільної бібліотеки проекту, що містить всі проектні робочі продукти, а також за відповідність робочих продуктів стандартам.

Методологія *RUP* використовує програмні рішення *Rational Application Developer*, *Rational Rose*, *Rational Asset Manager*, *Rational Software Architect*, *Rational Software Modeler*, *Rational Suite*, *Rational Requisitepro*, *Rational Clearquest*, *Rational Clearcase* та інші.

Переваги *RUP*:

- дуже широкий діапазон рішень в частині формалізації процесу розроблення;

- можна використовувати як основу для водоспадного стилю розроблення, так і в якості гнучкого процесу;

- зниження основних ризиків замовника і розробника;

- економія ресурсів за рахунок автоматизації регресійного тестування;

- поліпшення якості ПЗ за рахунок багаторазових перевірок змін;

- поліпшення якості тестування за рахунок використання сучасних технологій.

Недоліком є недостатній рівень формалізму, який може призводити до неузгодженості рішень, прийнятих учасниками проекту, до непродуктивних витрат ресурсів на переробку коду і на повторне рішення типових проблем.

1.2.2 Модель Microsoft Solutions Framework

У 1994 році *Microsoft* випустила *MSF* — набір концепцій і рекомендованих моделей, які дозволяють розробляти і впроваджувати розподілені інформаційні системи масштабу підприємства на основі технологій та інструментальних засобів фірми *Microsoft*. *MSF* базується на практичних результатах організації розподілених обчислень і застосування клієнт-серверних технологій, отриманих як у фірмі *Microsoft*, так і її партнерами і замовниками.

Процес розроблення відповідно до *MSF* є ітеративним і включає в себе такі основні фази: розробка концепції (єдиного бачення), планування, розробка, стабілізація (тестування), впровадження. Кожна фаза циклу закінчується головною віхою (контрольною точкою). Відповідно головні віхи будуть мати назви: концепція продукту затверджена, плани продукту затверджені, розробка завершена, готовність рішення затверджена, впровадження завершено. Віха є точкою синхронізації досягнутих результатів і очікувань замовника, а також аналізу проектного середовища. У рішенні про закриття чергової фази повинні приймати участь відповідальні представники всіх рольових кластерів (розробка, тестування, впровадження, керування проектом тощо).

MSF пропонує досить нестандартні підходи до організаційної структури, розподілу відповідальності і принципів взаємодії всередині команди. *MSF* пропонує розбиття великих команд (більше 10 осіб) на малі багатoproфільні групи напрямів (*feature teams*). Ці малі колективи працюють паралельно, регулярно синхронізуючи свої зусилля.

Microsoft пропонує інструменти для реалізації *MSF*: *MSF for Agile Software Development*, *MSF for CMMI Process Improvement*, *Visual Studio Team System (Team Suite; Team Edition for Database Professionals; Team Suite for Software Architects; Team Suite for Software Developers; Team Suite for Software Testers; Team Foundation Server; Team Test Load Agent)*.

Перевагами *MSF* є:

- систематизація та структуризація інформації у формі бази знань;

- нестандартні підходи до організаційної структури, розподілу відповідальності і принципів взаємодії всередині команди;
- легка масштабованість *MSF* — мінімальний розмір проектної групи в *MSF*-проекті 3 людини, але застосовувати дану методологію можна для колективів і в десятки, сотні, тисячі чоловік;
- не нав'язливість *MSF* щодо використання будь-яких конкретних інструментів і програмних засобів.

Недоліки *MSF*: не описує детально найважливіші ролі замовника і користувача; не розглядаються методи керування групою проектів.

1.2.3 Модель *Personal Software Process/Team Software Process*

В кінці 90-х років *SEI* розробив модель *PSP / TSP*, засновану на моделі зрілості *CMM* [12]. *PSP* визначає вимоги до компетенцій розробника. Відповідно до цієї моделі кожен програміст повинен вміти: враховувати час, витрачений на роботу над проектом; враховувати знайдені дефекти; класифікувати типи дефектів; оцінювати розмір задачі; здійснювати систематичний підхід до опису результатів тестування; планувати програмні завдання; розподіляти їх за часом і складати графік роботи; виконувати індивідуальну перевірку проекту та архітектури; здійснювати індивідуальну перевірку коду; виконувати регресійне тестування.

PSP включає 4 процеси, які послідовно додаються розробником: *PSP0* – базовий власний процес (The Baseline Personal Process); *PSP1* – планування (Personal Planning Process); *PSP2* – керування якістю (Personal Quality Management); *PSP3* – циклічний процес (Cyclic Personal Process).

TSP робить ставку на самокеровані команди чисельністю 3-20 розробників. Команди повинні встановити власні цілі, скласти свій процес і плани, відстежувати роботу, підтримувати мотивацію і максимальну продуктивність.

Методологія є ефективною, може забезпечити істотні переваги і стати моделлю для будь-якої організації, зацікавленої у вдосконаленні своїх процесів розроблення.

PSP / TSP — це чиста методологія щодо забезпечення якості проєктів, розрахована на використання *UML* при проєктуванні і будь-які *CASE*-засоби, зокрема вони орієнтовані на використання продуктів компанії *Microsoft*.

Переваги: найкраще планування часу та бюджету, керування якістю продукту, зменшення часу розроблення, акуратний збір метрик, оцінка часу роботи через оцінку обсягу артефактів, використання історичних даних у плануванні, раннє виявлення дефектів.

Недоліки: складність в обліку ризиків, обліку недооцінки складності.

1.2.4 Гнучка методологія розроблення програмного забезпечення (Agile)

У лютому 2001 року 17 розробників різних методологій зібралися для того, щоб спробувати виявити що-небудь спільне у своїх підходах. Результатом став Маніфест гнучкого розроблення. Тоді ж з'явився термін *Agile* («гнучкий, спритний»), що об'єднує всі методології.

Agile – гнучка методологія розроблення, це концептуальний каркас, в рамках якого виконується розробка програмного забезпечення. Основна ідея всіх гнучких моделей полягає в тому, що процес, який застосовується у розробці ПО, повинен бути адаптивним. Вони декларують своєю вищою цінністю орієнтованість на людей та їх взаємодію, а не на процеси і засоби.

CASE-засоби, що можуть допомогти впровадити і поліпшити процес створення ПЗ на базі *Agile*: *XPArchitect*; *Borland ALM* (зокрема *Silk*).

Розглянемо більш детально кілька гнучких методологій: *eXtreme Programming*, *Crystal*, *Feature Driven Development* та *Scrum*.

1.2.4.1 Екстремальне програмування eXtreme Programming

Екстремальне програмування (*eXtreme Programming*, або *XP* – методологія, розроблена Кентом Беком (*Kent Beck*), Уордом Каннінгамом (*Ward Cunningham*) і Роном Джеффріс (*Ron Jeffries*), є сьогодні найбільш відомою з гнучких методологій. *XP* проповідує комунікабельність, простоту, зворотний зв'язок. Вона описується як набір практик: гра в планування, короткі релізи, метафори, простий дизайн, переробки коду (*refactoring*), розробка «тестами вперед», парне програмування, колективне володіння кодом, 40-годинний робочий тиждень, постійна присутність замовника і стандарти коду. При використанні *XP* ретельне попереднє проектування ПО замінюється, з одного боку, постійною присутністю в команді замовника, а з іншого – регулярними переробками коду (так званий рефакторинг). Основою проектної документації вважається ретельно прокоментований код. Дуже велика увага в методології приділяється тестуванню.

Життєвий цикл проекту в *XP* складається з послідовності релізів. Кожен реліз – це повноцінна версія продукту, яку може використовувати замовник і яка містить додаткову функціональність у порівнянні з попереднім релізом. Реліз з'являється в результаті однієї або декількох ітерацій, що тривають від одного до чотирьох тижнів.

У *XP* не рекомендується витрачати багато часу на планування; сам процес планування називається грою (*planning game*). Докладний план складається тільки на чергову ітерацію і найближчі один-два релізи.

Планування релізу складається з наступних кроків:

- замовник формулює свої вимоги у вигляді історій, які оцінюються за трудомісткістю розробниками. Оцінки трудомісткості робляться в так званих ідеальних днях – часі, який якийсь уявний розробник витратить на реалізацію історії при повній відсутності відволікаючих чинників, паралельних завдань, перерв тощо;
- історії сортуються за пріоритетом, ризикам, складністю реалізації;
- визначається фактична продуктивність команди (яке число реальних днів відповідає ідеальному дню);

- на підставі фактичної продуктивності визначається, які історії увійдуть в черговий реліз і за який час він буде завершений.

Ітерація планується аналогічним чином:

- попередньо визначається набір історій для ітерації;
- історії розбиваються на завдання (tasks), які розподіляються між розробниками. На відміну від історій, які описують поведінку системи з точки зору користувача, завдання складаються на технічному рівні, наприклад «модифіковані схему бази даних» або «провести рефакторинг»;

- розробники оцінюють свої завдання. На цьому етапі уточнені оцінки повинні бути більш реалістичними. Оскільки завдання тепер оцінюють їхні безпосередні виконавці, в оцінках враховується індивідуальна продуктивність, що залежить від досвіду, знайомства з використовуваними технологіями тощо;

- на основі уточнених оцінок завдань підраховується загальне завантаження команди. Залежно від завантаження деякі історії можуть бути перенесені на наступну ітерацію або, навпаки, додані в поточну.

Періодично в ході проекту вимірюється фактична продуктивність команди. Якщо вона починає сильно відрізнятися від значення, яке було використано при плануванні, графік виходу релізів повинен бути переглянутий.

Недоліки:

- *XP* можуть ефективно використовувати тільки в команді досвідчених розробників;

- команду не можна розбивати на кілька частин, можливий розмір команди обмежений числом в 10-15 чоловік, тому що велику роль в *XP* грає пряме спілкування;

- не завжди можна забезпечити постійну присутність представника замовника у проектній команді;

- не бажано використовувати *XP* в проекті з фіксованою ціною.

Незважаючи на всі перераховані обмеження, *XP* може показати виняткову ефективність завдяки вкрай низьким накладним витратам.

1.2.4.2 Сімейство методологій Crystal

Crystal – сімейство методологій, що визначають необхідний ступінь формалізації процесу розроблення в залежності від кількості учасників і критичності завдань. Алістер Коуберн (*Alistair Cockburn*) на початку 90-х на замовлення від компанії IBM написав роботу на тему методології розроблення ПЗ. При цьому його підхід суттєво відрізнявся від підходу більшості інших методологів. Його теорії засновані не тільки на особистому досвіді, а й на постійних дослідженнях інших проектів і процесів.

Ступінь важливості нарастає по вертикальній осі. Величина команди нарастає по горизонтальній осі. У результаті виходить сімейство методологій. Чим нижче критичність і чим менше команда, тим більш «легку» методологію потрібно використовувати. Найлегшою з усього сімейства є методологія *Crystal Clear*. Головні принципи даної методології – вся команда розробників (до 6 осіб) знаходиться в одному приміщенні;

- часті поставки продукту дозволяють виробити «ритм» проекту;
- обмін інформацією з реальними користувачами дозволяє швидко отримувати зворотній зв'язок і ліквідувати недоліки на ранніх стадіях;
- засоби контролю версій коду забезпечують колективне володіння кодом.

Сімейство методологій *Crystal* побудовано на ітеративної розробці. Тривалість ітерації може змінюватися в межах від 1 до 4 місяців. Чим менше ітерація, тим краще.

Важливою особливістю *Crystal* є безперервне налагодження методології. Це дозволяє поступово адаптувати її для конкретної команди і конкретного проекту. В жодній іншій методології налаштуванню не приділяється такої уваги.

Перевагою *Crystal Clear* є те, що вона максимально проста у використанні, вимагає мінімальних зусиль для впровадження, орієнтована на людські звички, описує природний порядок розроблення ПЗ.

Недоліки:

- поступається *XP* за продуктивністю;

– дуже складно заздалегідь передбачити, які проміжні продукти необхідні.

1.2.4.3 Методологія Feature Driven Development

Методологія *Features Driven Development FDD* була розроблена Джеффом Де Люка (*Jeff De Luca*) і Пітером Коадам (*Peter Coad*) в 1997 році. Це функціонально-орієнтована розробка, яка оперує поняттям функції або властивості (feature) системи, досить близьким до поняття сценарію використання, вживаному в *RUP*. відмінність – це додаткове обмеження: «кожна функція повинна допускати реалізацію не більше ніж за два тижні».

FDD включає п'ять процесів, причому останні два повторюються для кожної функції: розробка загальної моделі; складання списку необхідних функцій системи; планування роботи над кожною функцією; проектування функції; конструювання функції. Робота над проектом передбачає часті збори і ділиться на ітерації, кожна з яких реалізується за допомогою певного набору функцій.

Переваги *FDD*: робить процес легше; дає можливість планування і попереднього проектування, створення детального дизайну, є керування пріоритетами і можливе трасування вимог; розрахований на роботу у великих командах.

Але в той же час, недолік документації може значно збільшувати вартість подальшого супроводу продукту, оскільки внесення будь-яких змін до нього вимагатиме дуже великих зусиль.

1.2.4.4 Методологія Scrum

Методологія *Scrum* була вперше озвучена в роботі Хіротака Такеучи й Ікуджіро Нонаки, опублікованій в *Harvard Business Review*. Джеф Сазерленд використовував цю роботу при створенні методології для корпорації Easel у 1993 році, яку за аналогією і назвав

Scrum, а Кен Швабер формалізував процес для використання в індустрії розроблення програмного забезпечення.

Мета цієї методології – виявити й усунути відхилення від бажаного результату на більш ранніх етапах розроблення програмного продукту.

Методологія *Scrum* визначає:

1. Правила, за якими повинен плануватися і управлятися список вимог до продукту, з метою досягнення максимальної прибутковості від реалізованої функціональності;
2. Правила планування ітерацій для забезпечення максимальної зацікавленості команди в отриманні результату;
3. Основні правила взаємодії учасників команди для максимально швидкої реакції на виникаючі робочі ситуації;
4. Правила аналізу і коригування процесу розроблення для вдосконалення взаємодії всередині команди.

Кожну ітерацію можна описати так: «Плануємо – Фіксуємо – Реалізуємо – Аналізуємо». За рахунок фіксування вимог на протязі однієї ітерації та зміни довжини ітерації можна керувати балансом між гнучкістю та плануємістю розроблення.

Scrum фокусується на постійному визначенні пріоритетних завдань, ґрунтуючись на бізнес цілях, що збільшує корисність і прибутковість проекту на його ранніх стадіях. Тому що при ініціації проекту його прибутковість визначити майже неможливо, *Scrum* пропонує концентруватися на якості розроблення і до кінця кожної ітерації мати проміжний продукт, який можна використовувати. Наприклад, результатом ітерації може бути каркас сайту, який можна показати на презентації.

Методологія *Scrum* орієнтована на те, щоб оперативно пристосовуватися до змін у вимогах, що дозволяє команді швидко адаптувати продукт до потреб замовника.

Девіз *Scrum* – «аналізуй та адаптуй»: аналізуйте те, що отримали, адаптуйте те, що вже розроблено до реальних потреб, а потім аналізуйте знову. Чим менше формалізму, тим більш гнучко та ефективно можна працювати, – це основний принцип даної методології. Але це не означає, що формальних процесів не повинно бути зовсім, їх має бути достатньо для організації ефективної взаємодії і керування проектом. Формальна частина *Scrum*

складається з трьох ролей, трьох практик і трьох основних документів.

В *Scrum*-проектах відділяють наступні ролі:

1. **Власник продукту** (*Product Owner*) – людина, що формулює вимоги програмістам. Зазвичай власник продукту є представником або довіреною особою замовника, а для компаній, що випускають коробкові продукти, він являє собою ринок, на якому реалізується продукт. Власник продукту повинен скласти бізнес план, який показує очікувану дохідність і план розвитку до вимог, відсортованими за коефіцієнтом окупності інвестицій. Виходячи з наявної інформації, власник продукту готує перелік вимог, відсортований за значимістю.

2. **Scrum-майстер** (*Scrum Master*) – забезпечує максимальну працездатність і продуктивність команди, чітку взаємодію між усіма учасниками проекту, своєчасне вирішення всіх проблем, що гальмують або зупиняють роботу будь-якого члена команди, відгороджує команду від усіх впливів ззовні під час ітерації та забезпечує проходження процесу всіх учасників проекту. Ця людина має бути одним з членів команди розроблення і брати участь у проекті як розробник. Він відповідає за своєчасне вирішення поточних проблем.

3. **Scrum-команда** (*Scrum Team*) – група, що складається з п'яти-дев'яти самостійних, ініціативних програмістів. Перше завдання цієї команди – поставити реально досяжну, прогнозовану, цікаву і значущу мету для ітерації. Друге завдання – зробити все для того, щоб ця мета була досягнута у відведені терміни і з заявленою якістю. Мета ітерації вважається досягнутою тільки в тому випадку, якщо всі поставлені завдання реалізовані, весь код написаний за певними проектом «стандартам кодування» (*coding guidelines*), програма протестована повністю, а всі знайдені дефекти усунені. Програмісти цієї команди повинні вміти оцінювати та планувати свою роботу, працювати в команді, постійно аналізувати та поліпшувати якість взаємодії та роботи. В обов'язки всіх членів *Scrum-команди* входить участь у виборі мети ітерації і визначення результату роботи. Вони повинні робити все можливе і неможливе для досягнення мети ітерації в рамках, визначених проектом, ефективно взаємодіяти з усіма учасниками команди, самостійно організовувати свою роботу, надавати власнику робочий продукт в кінці кожного циклу.

Scrum містить наступні документи: **журнал продукту** (*Product Backlog*), **журнал спринту** (*Sprint Backlog*) та **графік спринту** (*Burndown Chart*).

На початку проекту власник продукту готує **журнал продукту** (табл. 1.1) – перелік вимог, відсортований за значимістю, а *Scrum*-команда доповнює цей журнал оцінками вартості реалізації вимог. Перелік повинен містити функціональні та технічні вимоги, необхідні для реалізації продукту. Найбільш пріоритетні з них повинні бути досить детально прописані, щоб їх можна було оцінити і протестувати. Про своєчасну деталізацію вимог має дбати власник продукту і надавати необхідний обсяг у потрібний час. У цьому сенсі програмісти є замовниками вимог для власника продукту. Надалі інші вимоги повинні поступово уточнюватися та деталізуватися до такого ж рівня. Головне, щоб у команди завжди був достатній обсяг підготовлених до реалізації вимог.

Таблиця 1.1 – Приклад журналу продукту

Номер вимоги	Опис вимоги	Цінність бізнесу (ум. од.)	Пріоритет	Високорівнева оцінка(часи)
FE1	Покупець може зареєструватися на сайті	10.000	1	20
FE2	Покупець може ввести свої персональні дані	12.000	6	36
FE3	Покупець може побачити список доступних виробів	16.000	10	30
FE4	Покупець може купити виріб	100.000	20	48
FE5	Покупець може робити пошук виробів	80.000	30	32
FE6	Покупець може підписатись на новини	30.000	40	66

Після того як команда під час сесії планування вибрала і зобов'язалася реалізувати набір вимог з журналу продукту, ці вимоги розбиваються на більш дрібні завдання, що складають деталізований список вимог – **журнал спринту** (табл. 1.2). Розбивка на завдання повинна бути зроблена таким чином, щоб виконання однієї задачі займало не більше двох днів (вважається, що менш детальна, наприклад, півдня, розбивка призводить до більш грубої оцінки, ніж прийнятна в більшості проектів, що використовують методологію

Scrum). Розбивка на завдання допоможе так спланувати ітерацію, щоб в кінці не залишилося жодної невиконаної завдання і, відповідно, досягти її мети. Після завершення деталізації оцінка журналу спринту порівнюється з первинною оцінкою в журналі продукту. Якщо існує значна розбіжність, команда домовляється з власником продукту про обсяг робіт, який повинен бути виконаний протягом ітерації, і про те, який обсяг буде перенесено на наступну. Менш важливі і мало впливають на мету ітерації завдання виносяться з журналу спринту.

Графік спринту – показує щоденну зміну загального обсягу робіт, що залишилися до закінчення ітерації. Цей графік дозволяє команді розробників робити аналіз поточної ситуації та своєчасно реагувати на відхилення.

Таблиця 1.2 – Приклад журнал спринту

№ ви мо ги	№ зада-чі	Опис вимоги	Пріоритет	Високо-рівнева оцінка (час)	Оцінка (час)	Зал роб (час)
FE 1		Покупець може зареєструватися на сайті	1	20	24	19
	TA1	Розробити дизайн інтерфейсу користувача			1	0
	TA2	Закодувати інтерфейс Користувача			1	1
	TA3	Закодувати бізнес логіку та юніт тести			3	1
	TA4	Розробити структуру бази даних			1	1
	TA5	Закодувати таблиці та логіку бази даних			0,5	0,5
	TA6	Закодувати доступ до бази даних			2	2
	TA7	Задokumentувати класи			1	1
	TA8	Написати тести для інтерфейсу користувача			2	1
	TA9	Написати тести функціональності			2	1
	TA10	Автоматизувати тести			2	2
	TA11	Зарезервовано на огляд коду			2	2
	TA12	Зарезервовано на виправлення дефектів			2	2
	TA13	Зарезервовано на спілкування з власником продукту			1	1

	TA14	Зарезервовано на рефакторинг			2	2
	TA15	Зарезервовано на самовдосконалення			1	1
	TA16	Зарезервовано на аналіз дефектів			0,5	0,5

Графік спринту дозволяє також власнику продукту спостерігати за ходом ітерації – якщо загальний обсяг робіт не зменшується щодня, значить, щось йде не так. Під час сесії планування команда знаходить і оцінює завдання, які треба виконати для успішного завершення ітерації. Сума оцінок всіх завдань в журналі спринту є загальним обсягом роботи, який треба виконати за ітерацію. Після завершення кожного завдання Scrum-майстер перераховує обсяг роботи, що залишилася, і зазначає це на графіку спринту. Тільки в тому випадку, якщо по закінченні ітерації у журналі спринту не залишилося незавершених завдань, ітерація вважається успішною. Графік спринту використовується як допоміжний інструмент, що дозволяє коригувати роботу для завершення ітерації вчасно, з працюючим кодом і необхідною якістю.

К **Scrum-практикам** відносяться: спринт (*Sprint*), скрам (*Daily Scrum Meeting*) та демонстраційне засідання (*Sprint Review Meeting*).

1. Підготовка до першої ітерації, званої **спринт** (*Sprint*), починається після того, як власник продукту розробив **журнал продукту**.

При плануванні ітерації відбувається детальне розроблення сесій планування спринту (*Sprint Planning Meeting*), яке починається з того, що власник продукту, Scrum-команда і Scrum-майстер перевіряють план розвитку продукту, план релізу та перелік вимог, Scrum-команда перевіряє оцінки вимог, переконується, що вони досить точні, щоб почати працювати, вирішує, який обсяг роботи вона може успішно виконати за спринт, ґрунтуючись на розмірі команди, доступному часі та продуктивності. Scrum-команда повинна вибирати перші за пріоритетом вимоги з журналу продукту. Після того як Scrum-команда зобов'язується реалізувати обрані вимоги, Scrum-майстер починає планування спринту. Scrum-команда розбиває вибрані вимоги на завдання, необхідні для його реалізації. Ця активність в ідеалі не повинна займати більше чотирьох годин, і її

результатом є **журнал спринту**. Необхідно, щоб всі учасники команди взяли на себе зобов'язання з реалізації обраної мети.

2. Після закінчення планування починається ітерація. Кожен день Scrum-майстер проводить «скрам» (*Daily Scrum Meeting*) – п'ятнадцятихвилинної нарада, мета якої – досягти розуміння того, що сталося з часу попередньої наради, скоригувати робочий план до реалій сьогодення і позначити шляхи вирішення існуючих проблем. Кожен учасник Scrum-команди відповідає на три питання: що я зробив з часу попереднього скрама, що мене гальмує або зупиняє в роботі, що я буду робити до наступного скрама? У цій нараді може брати участь будь-яка зацікавлена особа, але тільки учасники Scrum-команди мають право приймати рішення. Правило обґрунтовано тим, що вони давали зобов'язання реалізувати мету ітерації, і тільки це дає впевненість у тому, що вона буде досягнута. На них лежить відповідальність за їх власні слова, і, якщо хтось з боку втручається і приймає рішення за них, тим самим він знімає відповідальність за результат з учасників команди.

В кінці кожного спринту проводиться **демонстраційне засідання** (*Sprint Review Meeting*) тривалістю не більше чотирьох годин. Спочатку Scrum-команда демонструє власнику продукту зроблену протягом спринту роботу, а той у свою чергу веде цю частину наради і може запросити до участі всіх зацікавлених осіб. Власник продукту визначає, які вимоги з журналу спринту були виконані, і обговорює з командою і замовниками, як краще розставити пріоритети в журналі продукту для наступної ітерації. У другій частині мітингу проводиться аналіз минулого спринту, який веде Scrum-майстер. Scrum-команда шукає використані в останньому спринті позитивні і негативні способи спільної роботи, аналізує їх, робить висновки і приймає важливі для подальшої роботи рішення. Scrum-команда також визначає програми, які можуть працювати краще, і шукає шляхи для збільшення ефективності подальшої роботи. Потім цикл замикається, і починається планування наступного спринту.

Час між ітераціями – це час прийняття основоположних рішень, що впливають на хід всього проекту. Під час спринту ніякі зміни ззовні не можуть бути зроблені. Після того як команда дала зобов'язання реалізувати журнал спринту, він фіксується, і зміни в ньому можуть бути зроблені тільки з таких причин:

- *Scrum*-команда протягом ітерації отримала кращу уяву про вимоги і потребує додаткових завдань для успішного завершення ітерації;

- знайдені дефекти, які треба обов'язково виправити для успішного завершення ітерації;

- *Scrum*-майстер і *Scrum*-команда можуть вирішити, що невеликі зміни, які не впливають на загальний обсяг робіт, можуть бути реалізовані в зв'язку з виниклою у власника продукту необхідністю.

Виходячи з того що журнал спринту не може бути змінений ззовні під час ітерації, потрібно вибирати його довжину, ґрунтуючись на стабільності вимог. Якщо вимоги стабільні, змінюються або доповнюються рідко, то можна вибрати шеститижневий цикл. У цьому випадку заощаджується час на перемикання команди з активного розроблення на планування та демонстраційні мітинги. Якщо вимоги часто змінюються і доповнюються, потрібно відштовхуватися від двотижневого циклу, в будь-якому випадку довжина ітерації – це величина експериментальна.

Недоліки *Scrum*:

1. Складно домогтися активної участі від кожного розробника і злагодженої колективної роботи в команді;

2. Складно залучити постачальника вимог до активної участі в проекті, зацікавити його динамікою розвитку продукту, дати можливість бути активним вболівальником і спонсором команди.

Проте, незважаючи на це, використання методології *Scrum* в проектах дозволяє:

- використовувати весь технічний багаж, накопичений компанією, тому що головний напрям методології *Scrum* направлений на керування проектами і не задає ніяких технічних практик;

- з високою якістю та в рамках бюджету реалізувати великий обсяг функціональності та специфікації, які були відсутні на момент початку проекту;

- забезпечити максимальну бізнес-цінність виробленого продукту, за рахунок того, що практично всі реалізовані функції активно використовуються відвідувачами;

- досягнути високої супровідності коду (можливість внесення змін з мінімальними трудовитратами) – вартість змін, внесених до

продукту, практично еквівалентна вартості розроблення аналогічних функцій продукту на початку проекту, що рідко буває у *RUP* чи *MSF* моделях виробництва, для яких характерний експонентний ріст вартості змін по мірі виконання проекту.

1.2.4.5 Методика Kanban

З початку 70-х років у Японії, а потім і в інших країнах набула великого поширення система «*Канбан*», що є механізмом організації безперервного гнучкого виробничого потоку і функціонує практично без страхових запасів. Традиційна концепція організації виробництва, як відомо, спрямована на запобігання простоям та організацію безперервного потоку з обов'язковим створенням страхового запасу. Японська концепція ґрунтується на практично повній відмові від страхових запасів. Більше того, менеджери навмисне дають робітникам змогу повністю випробувати на собі наслідки простоїв. В результаті весь персонал постійно зайнятий виявленням причин збоїв у виробництві та пошуком шляхів підвищення надійності та запасу міцності системи управління. Після виявлення та усунення причин простоїв керівники ще більше скорочують страховий запас, породжуючи додаткові зусилля з покращення організації виробництва з боку всього персоналу. В умовах системи «*Канбан*», на відміну від традиційного підходу, виробник не має завершеного плану й графіка виробництва, а жорстко пов'язаний конкретним замовленням споживача. Він не взагалі оптимізує свою роботу, а в межах замовлення. Конкретного графіка роботи на декаду, місяць він не має. Кожен зайнятий у технологічному ланцюгу робітник знає, що він вироблятиме продукцію тільки тоді, коли карта «*Канбан*» з його продукції відкріплена від контейнера на складі, тобто коли продукція фактично надійшла на наступну стадію обробки. Конкретний графік послідовності праці одержують лінії кінцевого складання, вони розкручують клубок інформації у зворотному напрямі. Іншими словами, графіки виробництва не переглядаються, а тільки формуються рухом карток «*Канбан*», тому що зняття карти відбору продукції, графіка виготовлення її фактично не було.

Виробництво постійно перебуває в стані надбудови, відбувається його системне настроювання під зміни ринкової кон'юнктури.

Методика розробки програмного забезпечення **Канбан** була введена у практичне використання Девідом Андерсеном у 2007 році. Багато з цих практик та підходів використовувалися різними *Agile* командами, перш ніж були описані як єдине ціле.

Нововведення полягали в тому що ,було введено завдання "в процесі". Це робилося і раніше іншими *Agile*-командами, але в Канбан існує всім відоме обмеження на кількість робочих завдань, які можуть виконуватися в один час. Ця межа зазвичай досить низька – ліміт приблизно дорівнює числу розробників в команді або трохи менший.

Основні положення *Канбан* наступні:

1. Візуалізація потоку робіт:

1.1 Розбиття роботи на частини, кожна частина виписується на картку і прикріплюється до стіни;

1.2 Розбиття усіх завдань на стовпчики для розділення по стадіям розробки;

2. Обмеження незавершеної роботи (НЗР) (*work-in-progress*) визначається можлива кількість незавершених пунктів на кожній стадії робочого процесу

3. Вимірювання часу виконання завдання (*lead time*) (середньої тривалості часу для завершення одного пункту, іноді звану "оперативним часом" (*cycle time*)), оптимізація процесу, щоб звести час виконання завдання до мінімуму і зробити його настільки прогнозованим, наскільки це можливо.

Канбан – це не конкретний процес, а система цінностей. Його можна описати однією простою фразою – «Зменшення виконується в певний момент роботи (*work in progress*)».

Різниця між *Канбан* і *Scrum*:

- В *Канбан* немає таймбоксів (ні на завдання, ні на спринти);
- В *Канбан* завдання більше і їх менше;
- В *Канбан* оцінки термінів на задачу опціональні або взагалі їх немає;
- В *Канбан* «швидкість роботи команди» відсутній і вважається тільки середній час на повну реалізацію завдання.

Команда для роботи використовує *Канбан*-дошку. Наприклад, вона може виглядати так (стовпці зліва направо):

Цілі проекту. Сюди можна помістити високорівневі цілі проекту.

Черга завдань. Тут зберігаються завдання, які готові до того, щоб почати їх виконувати. Завжди для виконання береться зверху, сама пріоритетна задача і її картка переміщується в наступний стовпець.

Опрацювання дизайну. Цей та інші стовпці до «Закінчено» можуть змінюватися, тому що саме команда вирішує, які кроки проходить завдання до стану «Закінчено». Наприклад, в цьому стовпці можуть перебувати завдання, для яких дизайн коду або інтерфейсу ще не ясний і обговорюється. Коли обговорення закінчені, завдання пересувається в наступний стовпець.

Розробка. Тут завдання перебуває до тих пір, поки розробка не завершена. Після завершення вона пересувається в наступний стовпець. Або, якщо архітектура не вірна або не точна – завдання можна повернути в попередній стовпець.

Тестування. У цьому стовпці завдання знаходиться, поки воно тестується. Якщо знайдені помилки – повертається в Розробку. Якщо ні – пересувається далі.

Деплоймент. У всіх проектів різне поняття деплоймент, наприклад, викласти нову версію продукту на сервер або помістити код в репозиторій.

Закінчено. Сюди стікер потрапляє лише тоді, коли всі роботи по завданню закінчені повністю.

В будь-якій роботі трапляються термінові завдання. Для таких можна виділити спеціальне місце (наприклад «Прискорити»). В **Прискорити** можна помістити один рядок до завдання і команда повинна почати її виконувати негайно і завершити якомога швидше. У такій черзі може бути тільки одна така задача, якщо з'являється ще одна – вона повинна бути додана в «Черга завдань».

Під кожним стовпцем вказуються цифри, які позначають кількість завдань, які можуть бути одночасно в цих стовпцях. Цифри підбираються експериментально, але вважається, що вони повинні залежати від числа розробників в команді.

Наприклад, якщо є 8 програмістів в команді, то в рядок «Розробка» можна помістити цифру 4. Це означає, що одночасно програмісти будуть робити не більше 4-х завдань, а значить у них буде багато причин для спілкування та обміну досвідом. Якщо

поставити туди цифру 2, то 8 програмістів, що займаються двома завданнями, можуть простоювати або втрачати занадто багато часу на обговореннях. Якщо поставити 8, то кожен буде займатися своїм завданням і деякі завдання будуть затримуватися на дошці надовго, а головне завдання *Канбан* – це зменшення часу проходження завдання від початку до стадії готовності.

Використання дошки дає такі переваги:

1. Зменшення числа паралельно виконуваних завдань сильно зменшує час виконання кожної окремої задачі, за рахунок того, що немає необхідності перемикаати контекст між завданнями, відстежувати різні сутності, планувати їх і т.д.

2. Легко побачити проблеми. Наприклад, якщо тестери не справляються з тестуванням, то вони дуже скоро заповнять весь свій стовпець і програмісти, які закінчили нову задачу, вже не зможуть перемістити її в стовпець тестування, тому що він заповнений. Для вирішення цієї проблеми, наприклад, програмісти можуть допомогти тестерам завершити одну із завдань тестування і тільки тоді пересунути нове завдання на звільнене місце. Це дозволить виконати обидва завдання швидше.

3. Можна обчислити час на виконання усередненої задачі. Можна позначати на картці дату, коли вона потрапила в чергу завдань, потім дату, коли її взяли в роботу і дату, коли її завершили. За цим трьом точкам для хоча б 10 завдань можна порахувати середній час очікування в чергу завдань і середній час виконання завдання.

Підсумовуючи, переваги *Канбан* полягають у тому, що:

1. Легко визначаються проблемні області;
2. Зростає продуктивність роботи за рахунок того, що члени команди техпідтримки самостійно організують свою роботу, використовуючи канбан-дошку, а менеджер лише спрямовує зусилля на пріорітезацію великих проектів і вирішення виникаючих проблем.

Недоліки *Канбан*:

1. Зі зменшенням НЗР з'являються обмеження – для менш пріоритетних проектів не вистачає ресурсів, що призводить до скорочення кількості проектів на команду.

2. *Канбан* накладає менше обмежень, ніж *Scrum*, тобто керівництво отримує більше параметрів для налаштування. Це може бути як недоліком, так і перевагою, залежно від ситуації.

3. *Канбан* – це ще більш «гнучка» методологія, ніж *Scrum* і *XP*, тому вона не підійде командам та проектам, не готовим до гнучкої роботи.

1.3 Практичні завдання

Завдання 1. Ви менеджер проекту «Система інтернет-банкінгу». Оберіть модель розробки, найбільш відповідну потребам вашого проекту. Обґрунтуйте свій вибір.

Завдання 2. Ви менеджер проекту «Симулятор водіння». Процес Визначте основні фази розроблення відповідно до *MSF*.

Завдання 3. Ви менеджер проекту «Гра-стратегія». Розпишіть план фаз і дисциплін ЖЦ проекту згідно *RUP*.

Завдання 4. Ви керівник проекту «Система керування проектами он-лайн». Наведіть процес тестування системи з використанням Канбан-дошки.

1.4 Контрольні запитання

1. Розкрийте поняття життєвий цикл. Перерахуйте можливі етапи розробки проекту.

2. Які принципи складають основу технологічних процесів в стандартах *ISO 9000:2000* та *ISO 15504-2-2009*?

3. Які основні цілі упорядкування, регламентування процесів та застосування стандартів у ЖЦ ПЗ?

4. Порівняйте моделі життєвого циклу програмних систем в *RUP* та *MFC*.

5. Порівняйте команди для *RUP* та *Scrum*.

6. Наведіть приклад використання моделі Канбан для компанії з тестування програмного забезпечення.

7. Які основні недоліки використання *RUP* роблять недоцільним його використання в невеликих проектних командах.
8. В чому особливість життєвого циклу проекту в *XP*?
9. Дайте опис процесу планування операції в *Scrum* проектах.

1.5 Література до розділу

Життєвий цикл ПЗ: [101-102],[115]; моделі програмних проектів: [49],[56],[59],[61], [68], [117].

2 ОРГАНІЗАЦІЯ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ТА СЕРТИФІКАЦІЯ ПРОГРАМНИХ ПРОДУКТІВ

2.1 Об'єкт, процес та система керування якістю програмних систем

Термін **якість** можливо розглядати з різних поглядів:

- з економічних трактувань **якість** – це задоволення очікувань споживача за ціну, яку він може собі дозволити, коли в нього виникає потреба, а висока якість – це перевищення очікувань споживача за нижчу ціну, ніж він припускає;

- *ISO* визначає **якість** як сукупність властивостей і характеристик продукту, які надають йому здатність задовольняти обумовлені чи передбачувані потреби;

- *IEEE* визначає **якість** як ступінь, в якій продукт володіє необхідною комбінацією властивостей.

Під керуванням якістю **SQM** (*Software Quality Management*) розуміють сукупність організаційної структури та відповідальних осіб, а також процедур, процесів і ресурсів для планування та керування досягненням якості програмних продуктів. **SQM** базується на застосуванні стандартних положень із забезпечення якості (ЗЯ) програмних систем **SQA** (*Software Quality Assurance*).

Головні терміни керування якістю:

- **рівень якості виробу** – це відносна оцінка якості, заснована на порівнянні сукупності характеристик виробу, відносно до базового;

- **властивість** – це об'єктивна здатність продукції, яка може виявлятися при її створенні, експлуатації та споживанні;

- **дефект** – це певна невідповідність продукції вимогам, встановленим нормативно-технічною документацією;

- **брак** – це дефектна одиниця продукції, тобто продукція, що має хоча б один дефект.

Мета процесу ЗЯ полягає в гарантуванні того, що продукти та процеси узгоджуються з вимогами, відповідним планом і містить наступні види діяльності:

- 1) впровадження стандартів і відповідних процедур розроблення програмних систем на всіх етапах життєвого циклу;

2) оцінювання дотримання цих стандартів і процедур.

До основних видів діяльності з гарантії якості відносяться:

- 1) перевірка несуперечності та виконання плану;
- 2) погодження проміжних програмних продуктів з плановими показниками;
- 3) перевірка виготовлених продуктів заданим вимогам;
- 4) аналіз проміжних процесів на відповідність договору і плану;
- 5) узгодження із замовником середовища та методів обробки продуктів;

6) перевірка прийнятих метрик, процесів, продуктів відповідно до затверджених стандартом і процедурами вимірами.

Мета процесу керування якістю – моніторинг якості для гарантії того, що продукт буде відповідати вимогам споживача.

Цей процес складається з:

- 1) визначення кількісних властивостей якості, заснованих на виявлених і передбачених потребах користувача;
- 2) керування реалізацією поставлених цілей для досягнення якості;
- 3) встановлення мети для досягнення необхідної якості.

Основні положення якості зі створення якісного продукту виділяють два процеси забезпечення якості:

- 1) підтвердження якості програмних систем як результату певної діяльності на кожному етапі життєвого циклу з перевіркою відповідності системи стандартам і процедурам, орієнтованих на досягнення якості;
- 2) інженерія якості як процес надання ПЗ властивостей функціональності, надійності, супроводу та інших характеристик якості.

Процеси досягнення якості призначені для:

- 1) керування розробленням та забезпечення гарантії відповідно до зазначених стандартів і процедур;
- 2) керування конфігурацією ризиків проекту відповідно до стандартів і процедур;
- 3) контроль базової версії програмної системи та реалізованих в ній характеристик якості.

Для виконання зазначених процесів виконуються наступні дії:

- 1) оцінювання стандартів і процедур, які виконуються при розробці програми;

- 2) реалізація керування розробленням і забезпечення гарантії якості ПЗ, а також проектної документації;
- 3) контроль проведення формальних інспекцій;
- 4) аналіз і контроль проведення приймального тестування.

Для організації, що займається керуванням якістю, інженерія якості повинна підтримуватися системою підтримки якості. Інженерія якості містить набір методів і заходів, за допомогою яких програмні продукти перевіряються на виконання вимог якості.

Система якості – це набір організаційних структур, методик, заходів, процесів і ресурсів для здійснення керування якістю.

Для забезпечення необхідного рівня якості ПЗ застосовуються два підходи.

У першому підході, орієнтованому на продукт, оцінка якості проводиться після випробувань. Цей підхід ґрунтується на припущенні, що чим більше виявлено та усунуто помилок при випробуванні, тим вища якість.

При іншому підході, орієнтованому на процес створення продукту, передбачаються та вживаються заходи щодо запобігання, випередження, виявлення та усунення помилок, починаючи з початкового етапу життєвого циклу відповідно до плану та процедур забезпечення якості розроблюваної системи.

Історію розвитку систем керування якістю наведено у додатку А.

Планування якості являє собою діяльність, спрямовану на визначення цілей і вимог якості, класифікацію та оцінку якості.

Оперативне керування містить методи та види діяльності оперативного характеру для поточного керування процесом проектування та усунення причин поганого або незадовільного функціонування програмної системи.

Забезпечення якості полягає у виконанні і перевірці того, що об'єкт розроблення виконує зазначені вимоги якісно.

Цілі забезпечення якості бувають внутрішні та зовнішні. Внутрішні спрямовані на створення впевненості керівника проекту в тому, що якість забезпечується. Зовнішні – на створення впевненості у користувача, що необхідну якість досягнуто й отримано якісне ПЗ.

Сучасна індустрія ПЗ характеризується дуже високим ступенем конкуренції. Для успішної роботи на цьому ринку компанія повинна розробляти, впроваджувати і супроводжувати програмне забезпечення

швидко, в строк і з задовільною якістю. Тому багато компаній вкладають гроші в поліпшення якості процесу, пам'ятаючи про те, що таке вкладення грошей обов'язково окупається – вивчення задокументованих випадків поліпшення процесів розроблення ПЗ показує, що в успішних випадках спостерігається істотне поліпшення продуктивності та якості з середнім рівнем повернення вкладень від 5:1 до 8:1.

2.2 Стандарти для систем керування якістю ПЗ

Існують десятки різних підходів до забезпечення якості ПЗ, і у всіх є свої переваги та недоліки. Розглянемо найбільш поширені та використовуємі стандарти, такі як *ISO 9001*, *CMM (Capability Maturity Model)* і *SPICE (ISO / IEC 15504)*.

Існують і інші досить розвинені стандарти, такі, як *Bootstrap*, багато в чому схожий з стандартами *CMM* і *SPICE*, *Trillium*, орієнтований на розробку продуктів в області телекомунікацій, *ISO 12207*, та інші.

2.2.1 Міжнародний стандарт ISO 9001

Однією з перших стандартів якості став стандарт *ISO (International Standard Organization – Міжнародної організації зі стандартизації)* серії 9000, перша версія якого була випущена в 1987 році. З тих пір сертифікати *ISO* серії 9000 зберігають незмінну популярність і визнаються у усьому світі.

Для процесу розроблення програм використовується стандарт *ISO 9001*, що передбачає проектування в процесі виробництва. Слід зазначити, що даний стандарт важко використовувати безпосередньо при керуванні якістю розроблення програмного забезпечення, оскільки спочатку він орієнтований на розробку промислових виробів. Спеціально для забезпечення процесів розроблення програмних систем організацією *ISO* розроблено посібник *ISO 9000-3*, який

формулює вимоги моделі якості *ISO 9001* до організації процесу розроблення програмного забезпечення.

Таким чином, для оцінювання якості процесу розроблення у власній організації або в організації підрядників можуть використовуватися вимоги керівництва *ISO 9000-3*. В даний час використовується версія стандарту 2008 року, який містить критерії для незалежної сертифікації системи керування якістю, дозволяючи продемонструвати зовнішнім зацікавленим сторонам гарантії мінімізації ризиків.

Стандарт *ISO 9001:2008* був прийнятий Міжнародною організацією зі стандартизації 14 листопада 2008.

ISO 9001 спрямований на застосування «процесного підходу» при розробці, впровадженні та поліпшенні результативності системи керування якістю для підвищення задоволеності споживачів шляхом виконання їхніх вимог.

В Україні наказом Держспоживстандарту України від 22.06.2009 № 225 затверджено національний стандарт *ДСТУ ISO 9001:2009* «Системи керування якістю. Вимоги» (*ISO 9001:2008, IDT*) з введенням в дію 01.09.2009.

Для результативного функціонування організація повинна визначити численні взаємопов'язані роботи та керувати ними. Роботу чи сукупність робіт, для якої використовують ресурси та якою керують для перетворення входів на виходи, можна вважати процесом. Часто вихід одного процесу безпосередньо є входом наступного.

Під «процесним підходом» розуміють застосовування в межах організації системи процесів разом з їх ідентифікуванням і взаємодіями, а також керуванням ними для одержання бажаного результату.

Перевага процесного підходу – в тому, що він забезпечує неперервний контроль зв'язків окремих процесів у межах системи процесів, а також їх з'єднань і взаємодій.

Застосовуючи такий підхід у межах системи керування якістю, особливу увагу приділяють:

- 1) розумінню та виконанню вимог;
- 2) потребі розглядати процеси з погляду створювання додаткових цінностей;

3) отриманню результатів функціонування процесу та досягненню результативності;

4) постійному поліпшенні процесів на основі об'єктивного вимірювання.

У зображеній на рисунку 2.1 моделі системи керування якістю в основу покладено процеси ЖЦ продукту та виявлені зв'язки між процесами. Замовники відіграють важливу роль у визначанні вимог, що на рисунку зображено як входи. Моніторинг задоволеності замовника вимагає оцінювання інформації, пов'язаної зі сприйняттям замовником того, як виконала організація його вимоги. Зображена на рисунку 2.1 модель охоплює всі вимоги цього стандарту, але не деталізує процеси.

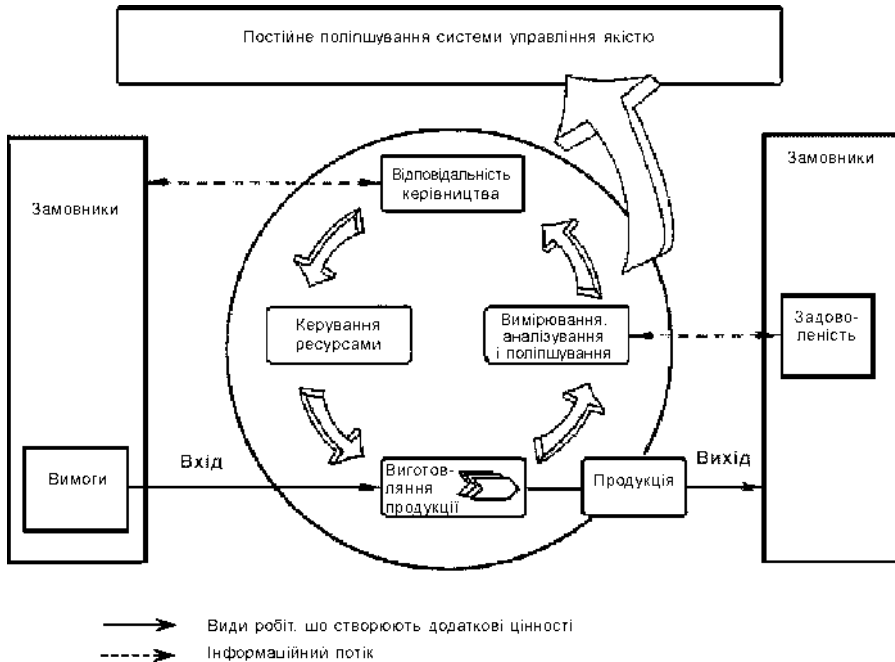


Рис.2.1. Модель системи керування якістю, в основу якої покладено процеси

На відміну від інших, стандарт *ISO 9001* має найбільшу спільність і зону застосування. Однак, *ISO 9001* охоплює вузьку область, залишаючи за рамками своєї уваги інші проблеми (не описуються зв'язок між організаційною підтримкою та реалізацією проекту, роль вимірювання в системі керування якістю, необхідність постійної оптимізації керування) процесу виробництва ПЗ та менеджменту цього процесу.

При застосуванні в системі менеджменту якості процесний підхід підкреслює важливість: розуміння та виконання вимог; необхідності розглядати процеси з точки зору доданої ними цінності; досягнення запланованих результатів виконання процесів і забезпечення їх результативності; постійного поліпшення процесів на основі об'єктивних вимірювань.

З точки зору багатьох західних і японських компаній, відповідність вимогам *ISO 9001* – це вкрай низький рівень гарантій якості, проте це той мінімальний рівень, який дає можливість входження в ринок.

2.2.2 Capability Maturity Model / Capability Maturity Model

Integration

У середині 80-х років XX століття на замовлення Міністерства оборони США Інститут програмної інженерії (*Software Engineering Institute – SEI*), що входить до складу Університету Карнегі-Меллона, розробив стандарт *SW-CMM (Capability Maturity Model For Software)* в якості еталонної моделі організації розроблення ПЗ при реалізації великомасштабних програмних проектів.

У моделі *CMM* визначено п'ять рівнів зрілості організацій. За результатами атестації компанії присвоюється певний рівень, який надалі може підвищуватися або (теоретично) знижуватися.

Початковий рівень (*initial level*) описаний в стандарті як основа для порівняння з наступними рівнями. На підприємстві з початковим рівнем організації не існує стабільних умов для створення якісного програмного забезпечення. Результат будь-якого проекту цілком і повністю залежить від особистих якостей менеджера та досвіду програмістів, до того ж успіх в одному проекті може бути

повторений тільки в разі призначення тих же менеджерів і програмістів на наступний проект. Більше того, якщо такі менеджери або програмісти йдуть з підприємства, то після їхнього звільнення різко знижується якість вироблених програмних продуктів. У стресових ситуаціях процес розроблення зводиться до написання коду і його мінімального тестування.

Для досягнення **повторюваного рівня** (*repeatable level*) на підприємстві повинні бути впроваджені технології керування проектами. При цьому планування та керування проектами ґрунтується на накопиченому досвіді, існують стандарти на ПЗ (причому забезпечується дотримання цих стандартів) і існує спеціальна група забезпечення якості. У разі необхідності організація може взаємодіяти з субпідрядниками. У критичних умовах процес має тенденцію знижуватися до початкового рівня.

Далі слідує **визначений рівень** (*defined level*), який характеризується тим, що стандартний процес створення та супроводження програмного забезпечення задокументований (зокрема і розроблення ПЗ і керування проектами). Мається на увазі, що в процесі стандартизації відбувається перехід на найбільш ефективні практики та технології. Для створення і підтримки такого стандарту в організації повинна бути створена спеціальна група. Нарешті, обов'язковою умовою для досягнення даного рівня є наявність на підприємстві програми постійного підвищення кваліфікації та навчання співробітників. Починаючи з цього рівня, організація перестає залежати від якостей конкретних розробників і не має тенденції скочуватися на рівень нижче в стресових ситуаціях.

На **керуваному рівні** (*managed level*) в організації встановлюються кількісні показники якості – як на програмні продукти, так і на процес в цілому. Таким чином, більш досконале керування проектами досягається за рахунок зменшення відхилень різних показників проекту. При цьому осмислені варіації в продуктивності процесу можна відрізнити від випадкових варіацій (шуму), особливо в добре освоєних областях.

Нарешті, **оптимізований рівень** (*optimizing level*) характеризується тим, що заходи щодо поліпшення застосовуються не тільки до існуючих процесів, але і для оцінки ефективності введення нових технологій. Основним завданням всієї організації на цьому рівні є постійне поліпшення існуючих процесів. При цьому поліпшення

процесів в ідеалі має допомагати попереджати можливі помилки або дефекти. Крім того, повинні вестися роботи по зменшенню вартості розроблення програмного забезпечення, наприклад, за допомогою створення та повторного використання компонент.

При сертифікації проводиться оцінка відповідності всіх ключових областей за 10-бальною шкалою. Для успішної кваліфікації певної ключовій області необхідно набрати не менше 6 балів.

Оцінювання виконується за наступними показниками:

1. Зацікавленість керівництва (чи планується практичне впровадження даної ключовій області, чи існує розуміння у керівництва необхідності даної області тощо).

2. Широта використання даної області в організації (наприклад, оцінці в 4 бали відповідає фрагментарне застосування).

3. Успішність використання сертифікуємої області на практиці (наприклад, оцінці в 0 балів відповідає повна відсутність будь-якого ефекту, а оцінка у 8 балів виставляється за наявності систематичного і вимірного позитивного результату практично у всій організації).

Ключові області процесу описуються за допомогою наборів ключових практик. Існує наступна класифікація ключових практик: зобов'язання (*commitments to perform*), можливості (*abilities to perform*), діяльності (*activities performed*), виміри (*measurements and analysis*) та перевірки (*verifying implementations*).

Наприклад, керування вимогами пов'язано з наступними практиками:

1. Зобов'язання. Проекти повинні дотримуватись певної політики організації з керування вимогами.

2. Можливості. У кожному проекті має визначатися відповідальний за аналіз системних вимог і прив'язку їх до апаратного, програмного забезпечення та інших компонентів системи. Вимоги повинні бути задокументовані. Для керування вимогами повинні бути виділені адекватні ресурси та бюджет. Персонал повинен проходити навчання в області керування вимогами.

3. Діяльності. Перш ніж бути занесеними до проекту, вимоги піддаються аналізу на повноту, адекватність, несуперечність та ін. Виділені вимоги використовуються як основа для планування та виконання інших робіт. Зміни у вимогах аналізуються і додаються до проекту.

4. Виміри. Проводиться періодичне визначення статусу вимог та статусу діяльності з керування ними.

5. Перевірки. Діяльність з керування вимогами періодично аналізується старшими менеджерами. Діяльність з керування вимогами періодично і на підставі значущих подій аналізується менеджером проекту. Група забезпечення якості проводить аналіз і аудит діяльності з керування вимогами і звітує за результатами цього аналізу.

В таблиці 2.1 наведений приклад оцінювання практик різних видів, приписаних до різних ключових областей процесу.

Таблиця 2.1. Кількість ключових практик в різних областях процесу відповідно до СММ версії 1.1

Рівні	Область процесу	Обов'язки	Можливості	Діяльність	Виміри	Перевірки
2	Керування вимогами	1	4	3	1	3
	Планування проектів	2	4	15	1	3
	Моніторинг проекту	2	5	13	1	3
	Керування підрядниками	2	3	13	1	3
	Забезпечення якості ПЗ	1	4	8	1	3
	Керування конфігурацією	1	5	10	1	4
3	Контроль дотримання технологічного процесу	3	4	7	1	1
	Вироблення і підтримка технологічного процесу	1	2	6	1	1
	Навчання персоналу	1	4	6	2	3
	Інтегроване керування	1	3	11	1	3
	Розробка програмного продукту	1	4	10	2	3
	Координація діяльності груп	1	5	7	1	3
	Проведення експертиз	1	3	3	1	1
4	Керування процесом на основі метрик	2	5	7	1	3
	Керування якістю ПЗ	1	3	5	1	3
5	Запобігання дефектів	2	4	8	1	3
	Керування змінами технологій	3	5	8	1	2
	Керування змінами процесу	2	4	10	1	2

У результаті атестації компанії присвоюється певний рівень, який надалі може підвищуватися або (теоретично) знижуватися. Даний стандарт передбачає можливість сертифікації лише одного процесу або підрозділу організації.

Недоліки *CMM*:

- стандарт *CMM* є власністю *SEI* і не є загальнодоступним;
- оцінка якості процесів організацій може проводитися тільки спеціалістами, які пройшли спеціальне навчання і акредитовання *SEI*;
- стандарт орієнтований на застосування у відносно великих компаніях;
- стандарт перебільшує роль формальних описів процесів;
- в стандарті не розглянуті питання про відбір, підвищення кваліфікації та збереження компетентних співробітників.

Модель зрілості *Capability Maturity Model Integration (CMMI)*.

Після появи *CMM* стали розроблятися спеціалізовані моделі зрілості для створення інформаційних систем, для процесу вибору постачальників і деякі інші. На їх основі була розроблена інтегрована модель *CMMI*.

CMMI уточнює та удосконалює попередні моделі *CMM*, а також враховує основні вимоги існуючих міжнародних стандартів у галузі менеджменту програмних засобів. У 2006 року випустили версію *CMMI* – 1.2, яка є модернізацією версії *CMMI* – 1.1. Діюча версія має номер 1.3 (вийшла в листопаді 2010 року).

Основні зміни в порівнянні з *CMM* наступні:

1. Створено дві дещо відмінні за викладом моделі – безперервна і поетапна. Перша призначена для полегшення міграції від підтримки американського галузевого стандарту *EIA/AIS 713* [3.1] і поступового вдосконалення процесів за рахунок впровадження різних практик. Інша призначена полегшенню міграції від підтримки *CMM* і поетапного перегляду введених практик.

2. Елементи моделі отримали чітке визначення чи є вони обов'язковими (*required*), рекомендованими (*expected*) або інформативними (*informative*).

3. Практики поділяються на загальні (*generic*) і специфічні (*specific*). Вони доповнюються набором загальних і специфічних цілей, які необхідні для досягнення певного рівня зрілості в обраних областях процесу.

4. Деякі рівні зрілості отримали інші назви. Другий рівень названий керованим (*managed*), а четвертий – керованим на базі кількісної інформації (*quantitatively managed*).

5. Набір виділених областей процесу та практик значно змінився.

Всі області процесу діляться на 4 категорії. У наведеній нижче таблиці 2.2 вказано рівень, починаючи з якого вони повинні підтримуватися згідно *CMMI*, та визначена область процесу.

Таблиця 2.2. Розподіл практик за рівнем та областю процесу

Рівень CMMI Область процесу	2-ий рівень	3-ій рівень	4-ий рівень	5-ий рівень
Керування процесом		вироблення і підтримка процесу, контроль дотримання процесу, навчання	процеси 3-ого рівня + вимірювання показників процесу	процеси 4-ого рівня + впровадження інновацій
Керування проектом	планування проектів, контроль ходу проекту, керування угодами з постачальниками	процеси 2-ого рівня +, інтегроване керування проектами, керування ризиками, побудова команд, керування постачальниками	процеси 3-ого рівня + вимірювання показників результативності та ходу проекту	
Технічні процеси	керування вимогами	процеси 2-ого рівня + вироблення вимог, вироблення технічних рішень, інтеграція продуктів, верифікація і валідація		
Підтримуючі процеси	керування конфігурацією забезпечення якості продуктів і процесів, проведення вимірювань та аналіз їх результатів	процеси 2-ого рівня + керування оточенням, аналіз та прийняття рішень	процеси +3-ого рівня	процеси +4-ого рівня + аналіз, дозвіл і запобігання проблем

Керування процесом – містить вироблення і підтримку процесу, контроль дотримання процесу, навчання, вимірювання показників процесу, впровадження інновацій (5).

Керування проектом – містить планування проектів, контроль за ходом проекту, керування угодами з постачальниками, інтегроване керування проектами, керування ризиками, побудова команд, керування постачальниками та вимірювання показників результативності та ходу проекту.

Технічні процеси – містять формулювання вимог, керування вимогами, вироблення технічних рішень, інтеграцію продуктів, верифікацію і валідацію.

Підтримуючі процеси – містять керування конфігурацією, забезпечення якості продуктів і процесів, проведення вимірювань та аналіз їх результатів, керування оточенням, аналіз та прийняття рішень, аналіз, дозвіл і запобігання проблем.

Значна увага в *СММІ* приділяється процесам розроблення та обліку ітерацій при зміні вимог замовників, їх відстеження до функцій, компонентів, тестів і документів проекту.

Недоліком є те, що, так як модель *СММІ* передбачає суворе і формальне планування, відстеження проекту і повне документування, іншими словами, великі накладні витрати, через це ці моделі недоцільно застосовувати до розробок невеликих систем в рамках маленьких проектів з невеликою кількістю співробітників, а також проектів з мінливими вимогами або з короткими циклами розроблення.

Моделі *СММ* / *СММІ* підходять для довгих проектів (програм) і великих організацій, які створюють складні багатокомпонентні системи з поставками нових версій кожні 9-12 місяців і відносно великим числом співробітників.

2.2.3 Стандарт версії ISO/IEC 15504

У 1991 році Міжнародна організація з стандартизації ініціювала роботу зі створення єдиного стандарту оцінки програмних процесів. Стандарт отримав ім'я *SPICE* (скорочення від *Software Process Improvement and Capability dEtermination*, визначення можливостей і

поліпшення процесу створення програмного забезпечення). Офіційно стандарт називається «*ISO / IEC 15504: Information Technology – Process Assessment*». Міжнародний стандарт версії *ISO/IEC 15504* в даний час містить 6 частин, а 7-а частина зараз знаходиться в остаточній формі та розпочато роботу над 8-ою частиною.

ISO/IEC 15504 – є еталонною моделлю, що визначає такі якості процесу як «Вимірювання процесу» і «Вимірювання зрілості (можливостей)».

«Вимірювання процесу» поділяє процеси на п'ять категорій: відношення клієнт – постачальник; інженерні процеси; процеси підтримки; процеси керування; організаційні кроки. «Вимірювання зрілості (можливостей)» визначається за допомогою дев'яти атрибутів процесу, згрупованих на 5 рівнях зрілості:

- 1.1 Уява процесу (*Process Performance*);
- 2.1 Керування процесом (*Performance Management*);
- 2.2 Робота з керування продуктами (*Work Product Management*);
- 3.1 Визначення процесу (*Process Definition*);
- 3.2 Розгортання процесу (*Process Deployment*);
- 4.1 Вимірювання процесу (*Process Measurement*);
- 4.2 Контроль процесу (*Process Control*);
- 5.1 Інновація процесу (*Process Innovation*);
- 5.2 Оптимізація процесу (*Process Optimization*).

У таблиці 2.3 наведено список рівнів здібностей моделі SPICE і характерні для них процедури керування. Зазначимо, що на даний момент не існує українського перекладу стандарту SPICE, тому використані терміни не є загальноприйнятими або офіційно зареєстрованими.

ISO/IEC 15504 може використовуватися в двох контекстах: для вдосконалення якості процесів розроблення і для визначення якості.

Кожен атрибут процесу оцінюється за шкалою: не досягнуто (0 – 15%); частково досягнуто (15 – 50%); в основному досягнуто (50 – 85%); повністю досягнуто (85 – 100%).

Оцінювання процесу відбувається шляхом порівняння процесу розроблення ПЗ, що існує в певній організації, з описаною в стандарті моделлю.

Таблиця 2.3. Рівні здібностей процесу, що стандартизується в стандарті SPICE

Рівні	Назва
Рівень 0	Процес не виконується
Рівень 1	Процес здійснюється
1.1	Вимірювання продуктивності процесу
Рівень 2	Керований процес
2.1	Керування продуктивністю
2.2	Керування створенням продуктів
Рівень 3	Встановлений процес
3.1	Документування процесу
3.2	Відстеження ресурсів процесу
Рівень 4	Передбачуваний процес
4.1	Вимірювання процесу
4.2	Керування процесом
Рівень 5	Оптимізуючий процес
5.1	Зміна процесу
5.2	Постійне вдосконалення

Аналіз результатів, отриманих на цьому етапі, допомагає визначити сильні та слабкі сторони процесу, а також внутрішні ризики, властиві даному процесу. Це допомагає оцінити ефективність процесів, визначити причини погіршення якості та пов'язані з цим витрати в часі або оцінити можливості поліпшення цього процесу. Дуже часто визначення можливостей процесу здійснюється компанією-постачальником, щоб переконати існуючих або потенційних замовників у своїй здатності досягти заданих показників.

В результаті попередніх кроків, в організації може з'явитися розуміння необхідності поліпшення того чи іншого процесу. До цього моменту мета вдосконалення процесу вже чітко сформульована і залишається тільки технічна реалізація поставлених завдань. Після цього весь цикл робіт починається спочатку.

Одним з важливих переваг *SPICE* є його відкритість і вільне розповсюдження.

У таблиці 2.4 коротко сформульовані переваги *SPICE* в порівнянні з *ISO 9001*:

Таблиця 2.4. Порівняння стандартів *SPICE* та *ISO 9001*

<i>SPICE</i>	<i>ISO 9001</i>
Об'ємний і детальний документ	Короткий документ
Детальна модель	Абстрактна модель
Поліпшення процесу і визначення можливостей	Тільки сертифікація
Шість рівнів можливостей процесів	Сертифікація / відмова
Вимоги до оцінки процесу, настанови щодо застосовування	Тільки модель
Доповнює <i>ISO 9001</i>	Може бути деталізований за допомогою <i>SPICE</i>

SPICE надає більш повний набір засобів щодо забезпечення якості та поліпшення процесів, ніж *ISO 9001*. Тому для забезпечення якості процесів розроблення ПЗ рекомендують використовувати саме *SPICE*. Це допоможе організації помітно поліпшити існуючі процеси, а потім при необхідності отримати і сертифікат *ISO 9001*. Використовувати *SPICE* можна і в невеликих компаніях – про це свідчать результати роботи проекту *SPIRE*, в рамках якого проводилося впровадження процесів поліпшення якості в малих (менше 50 чоловік) компаніях з різних країн Європи. Як показав цей досвід і при невеликих грошових вкладеннях в маленьких компаніях можна добитися істотного збільшення продуктивності праці і поліпшення якості вироблених продуктів.

Недоліки *SPICE* :

– *ISO / IEC 15504* не є настільки успішним, як *CMM*, так як *CMM* активніше спонсується, був створений раніше і згодом був замінений *CMMI*, яка включає в себе безліч ідей *ISO / IEC 15504*, а також зберігає переваги *CMM*;

– відповідно до *ISO / IEC 15504* не видаються сертифікати відповідності. Він не є нормативним, хоча відповідає сучасним міжнародним вимогам до якості.

2.3 Сертифікація програмних продуктів

2.3.1 Процес сертифікації програмних продуктів

Сертифікація складається з ряду організаційних процесів, що становлять **систему сертифікації**, ці процеси підтримуються регламентованими процедурами і документами та повинні виконуватися кваліфікованими, атестованими експертами – інспекторами.

Для сертифікації підприємства розробника і результатів його діяльності – програмних продуктів, моделями CMMI або профілями стандартів ISO рекомендується певна дисципліна, що повинна бути адаптована до конкретних характеристик об'єктів і зовнішнього середовища життєвого циклу ПЗ. Наведені нижче процеси і документи орієнтовані на великі проекти, і їх склад може скорочуватися за погодженням між розробниками, замовниками та сертифікатором в простіших випадках.

Роботи з сертифікації починаються з акредитації органу або випробувальної лабораторії, формування та подання до Центрального органу з сертифікації заявки і комплекту документів для прийняття рішення про доцільність акредитації. При позитивних результатах перевірки оформляється та видається атестат акредитації.

Положення про орган сертифікації або лабораторії є основним документом, що встановлює тематичну область акредитації, юридичний статус, функції, структуру, права та обов'язки, методи, засоби і організацію випробувань. Паспорт сертифікаційної лабораторії (центру) повинен містити відомості про оснащеність засобами обчислювальної техніки, необхідними для проведення випробувань, про персонал і кадровий склад, оснащеність інструментальними засобами проведення випробувань, забезпеченість нормативними, технічними та методичними документами, а також іншими ресурсами, необхідними для випробувань.

Настанова з якості містить виклад принципів, опис методів і процедур, пов'язаних з виконанням основних функцій та завдань органу з сертифікації або лабораторії, які забезпечують якість

проведених випробувань і довіру до результатів оцінок, випробувань та експертиз.

Керівництво з якості, як правило, містить розділи:

- політика в галузі забезпечення якості проведення випробувань та експертиз;
- оснащення центру актуальними методологічними матеріалами і програмно-інструментальними засобами випробувань;
- формалізація вимог до об'єктів випробувань;
- політика в області технічної оснащеності центру та підвищення кваліфікації персоналу;
- архівація та контроль збереження документації результатів сертифікації.

Заявник для оцінювання продукції або процесу, що підлягають сертифікації, направляє до органу з сертифікації заявку за формою, прийнятої в системі сертифікації. Орган з сертифікації проводить роботу з підготовки та організації сертифікації продукції за заявкою. Ця робота містить:

- вибір схеми сертифікації з урахуванням специфіки продукції (обсяг, технологія, вимоги нормативних документів тощо) та пропозицій розробника;
- визначення кількості та порядку відбору зразків та компонентів, що підлягають випробуванням, якщо це не зазначено в стандартах;
- вибір і визначення акредитованої випробувальної лабораторії, яка повинна проводити випробування;
- підготовку проекту договору на виконання робіт.

Підготовча частина роботи з сертифікації закінчується випуском рішення за формою, прийнятої в системі сертифікації. Рішення разом з проектами договору на виконання робіт надсилається заявнику. При організації сертифікаційних випробувань здійснюється підбір і вивчення діючих нормативних документів на продукцію, заявлену до сертифікації, методів її випробувань і оцінки результатів.

Заявник приймає остаточні рішення, які елементи системи якості, дільниці та види організаційної та технічної діяльності підлягають перевірці під час сертифікації в заданий інтервал часу. Заявник повинен створити умови і надати документи для забезпечення процесів перевірок. Він може подати до органу з сертифікації протоколи випробувань, проведених при розробці та постановці

продукції на виробництво, документи про випробування, виконаних сторонніми випробувальними лабораторіями та інші документи, що свідчать про відповідність технології чи продукції встановленим вимогам. На основі аналізу поданих із заявкою документально підтверджених доказів відповідності його продукції встановленим вимогам, орган з сертифікації може прийняти рішення про скорочення обсягу випробувань або про видачу сертифіката.

Випробування проводяться випробувальними лабораторіями, акредитованими на проведення тільки тих випробувань, які передбачені в їх нормативних, акредитаційних документах. При неможливості проведення випробувань на випробувальній базі акредитованої лабораторії, випробування можуть проводитися персоналом цієї лабораторії у виробника чи споживача даної продукції з використанням власних коштів випробувальної лабораторії чи наявних у постачальника засобів випробувань.

Процес сертифікації програмних продуктів та систем якості підприємства містить:

- аналіз і вибір розробником або замовником (заявником) компетентних у цій галузі органу і атестованої лабораторії для виконання сертифікаційних випробувань;

- подачу заявником заявки на випробування до органу сертифікації і прийняття сертифікатора рішення за заявкою, вибір схеми сертифікації, укладання договору на сертифікацію;

- ідентифікацію вимог до системи якості підприємства та/або до версії програмного продукту, що підлягають випробуванням;

- виконання сертифікаційних випробувань системи якості підприємства або версії програмного продукту сертифікаційної лабораторією;

- аналіз отриманих результатів і прийняття рішення лабораторією та/або органом сертифікації про можливість видачі заявнику сертифіката відповідності;

- видачу органом сертифікації заявнику – сертифікату та ліцензії на застосування знака відповідності та на випуск сертифікованої продукції – версій програмного продукту;

- здійснення інспекційного контролю органом сертифікації сертифікованої системи якості підприємства та/або продукції;

- проведення заявником коригувальних заходів у разі порушення відповідності процесів системи якості та/або продукції

встановленим вимогам і при неправильному застосуванні знака відповідності.

При перевірці відповідальності керівництва розробника за якість продукції повинно бути визначено наявність у підприємства або проекту, документально оформлених політики, цілей і зобов'язань у сфері якості, а також ступінь розуміння цієї політики, її практичне здійснення і підтримка в робочому стані на всіх рівнях організації. Повинно бути встановлено наявність на підприємстві представника керівництва, який незалежно від інших обов'язків має повноваження і несе відповідальність за постійне виконання вимог стандартів і нормативних документів системи якості. Слід перевіряти наявність вимог, процедур, засобів і навченого персоналу для практичної реалізації процесів системи якості, а також актуальність і систематичність оформлення документації на всі компоненти, вимоги та положення системи якості, що представляє собою інтегрований процес протягом усього життєвого циклу ПЗ. Перевірки системи якості повинні включати визначення:

- наявності та повноти технологічної документації та дотримання її вимог на практиці;
- стану засобів технологічного оснащення та наявності системи їх технічного обслуговування;
- наявності та ефективності системи контролю та випробувань;
- стану засобів вимірювань і випробувань;
- наявність системи виявлення та усунення виявлених недоліків продукції або технології.

На підставі випробувань оцінюються отримані результати і обґрунтовуються висновки про відповідність або невідповідність продукції або процесів вимогам нормативних документів. Протоколи випробувань подаються до органу з сертифікації, а також заявнику на його вимогу. Протоколи випробувань підлягають зберіганню протягом строків, встановлених у правилах систем сертифікації продукції і в документах випробувальної лабораторії, але не менше трьох років.

Після отримання та перевірки комплектності і якості документації фахівцями випробувальної лабораторії слід провести експертизу ступеня реального застосування системи якості на підприємстві. Випробування починаються з складання **програми**

перевірки системи якості, яка повинна служити робочим планом проведення подальших робіт.

Програма є внутрішнім робочим документом випробувальної лабораторії і повинна містити перелік робіт, деталізовану відповідно до специфіки підприємства-розробника і містить аналіз повноти та якості поданих вихідних документів і ступеня їх практичного застосування при проектуванні, розробленні та постачанні ПЗ. Експертиза застосування процедур системи якості здійснюється випробувальною лабораторією на робочих місцях підприємства, що забезпечує ЖЦ ПЗ. Перевірки виконуються за наявності на робочих місцях спеціалістів – розробників відповідних документів та за повнотою використання їх положень та рекомендацій. Аналізи стану проекту і внутрішні перевірки системи якості, процесів та/або продукції повинні виконуватися персоналом, незалежним від осіб, безпосередньо відповідальних за виконання цих робіт.

Методики перевірок якості розроблення ПЗ повинні бути забезпечені необхідними ресурсами для виконання програми випробувань, методиками планування та розроблення приватних процедур перевірок. Методики повинні містити: об'єкти і цілі випробувань; показники якості, що будуть оцінюватися; умови і порядок випробувань; методи оброблення, аналізу і оцінки результатів випробувань; технічне забезпечення випробувань і звітність. Слід вказувати технічні й програмні засоби, що використовуються під час проведення випробувань, і порядок проведення випробувань, а також очікувані результати перевірок. Повинні бути розроблені методики контролю за результатами корекцій, дій по виправленню дефектів, якщо в службу керування перевірок надійде такий запит. Служба керування програмами випробувань повинна розробити методики збереження конфіденційності будь-якої інформації про випробування, а також даних, наявних у експертів.

Протоколи випробувань видаються заявнику до органу з сертифікації. Заявник може подати до органу з сертифікації протоколи випробувань з урахуванням термінів їх дії, проведених при розробці та постановці продукції на виробництво, або документи про випробування, виконаних вітчизняними або закордонними випробувальними лабораторіями, акредитованими або визнаними в системі сертифікації. На підставі протоколів сертифікаційних випробувань оцінюються отримані результати і виконується

обґрунтування зроблених висновків про відповідність або невідповідність продукції вимогам нормативних документів.

Висновок за результатами сертифікаційних випробувань розробляється представником органу із сертифікації та містить узагальнені відомості про результати випробувань та обґрунтування доцільності видання сертифіката. У разі отримання негативних результатів сертифікаційних випробувань приймається рішення про відмову у виданні сертифікату відповідності. Після доопрацювання система якості сертифікується або випробування можуть бути повторені. Результати аналізу стану технології та якості продукції оформляються актом, в якому надаються оцінки по всіх позиціях Програми випробувань і містяться висновки, що містять загальну оцінку стану виробництва та продукції, необхідність коригувальних заходів. Акт використовується органом по сертифікації разом з протоколами випробувань, заявкою на видачу та визначення терміну дії сертифіката на програмний продукт, періодичності інспекційного контролю, а також для складання коригуючих заходів.

За результатами сертифікаційних випробувань та експертизи документації приймається рішення про видачу сертифіката. У разі отримання негативних результатів сертифікаційних випробувань приймається рішення про відмову у видачі сертифіката відповідності. Крім того, підприємству-заявнику може бути направлено пропозиції щодо усунення можливих причин негативних результатів випробувань, після доопрацювання сертифікується випробування можуть бути повторені.

Орган з сертифікації після аналізу протоколів випробувань, оцінки виробництва, сертифікації системи якості, аналізу документації, зазначеної в рішенні за заявкою, здійснює оцінку відповідності продукції встановленим вимогам, оформляє сертифікат на підставі висновку експертів і реєструє його. При внесенні змін в конструкторську або експлуатаційну документацію, які можуть вплинути на якість системи або програмний продукт, що засвідчуються при сертифікації, заявник повинен сповістити про це орган з сертифікації, для прийняття рішення про необхідність проведення додаткових випробувань. Після реєстрації сертифікат вступає в силу і прямує підприємству-заявнику. Одночасно з видачею сертифіката підприємству-заявнику може видаватися ліцензія на право застосування знака відповідності.

За сертифікованими програмними продуктами в процесі їх експлуатації протягом усього терміну дії сертифіката відповідності повинен здійснюватися **інспекційний контроль**. Інспекційний контроль проводиться у формі періодичних і позапланових перевірок дотримання вимог до якості технології та сертифікованої продукції.

Об'єктами контролю, в залежності від схеми сертифікації, є сертифікована продукція, система якості або стабільність виробництва підприємству-заявнику. При визначенні періодичності і обсягу інспекційної перевірки враховуються такі фактори: ступінь потенційної небезпеки програмного продукту, стабільність виробництва, обсяг випуску, наявність і застосування системи якості при розробці, інформацію про результати випробувань продукту і його виробництва, проведених виробником, органами державного контролю і нагляду.

Результати інспекційного контролю оформляються актом, в якому дається оцінка результатів випробувань зразків та інших перевірок, робиться загальний висновок про стан виробництва сертифікованої продукції і можливості збереження дії виданого сертифіката. Акт зберігається в органі по сертифікації, а його копії направляються розробнику і в організації, що брали участь в інспекційному контролі. За результатами інспекційного контролю орган по сертифікації може призупинити або скасувати дію сертифіката й анулювати ліцензію на право застосування знака відповідності у разі невідповідності продукції вимогам нормативних документів, контрольованих при сертифікації, а також у випадках:

- принципів змін моделі зрілості, профілю стандартів, нормативних документів на продукцію або методу випробувань;
- зміни конструкції (складу), комплектності продукції;
- зміни організації або технології розроблення та виробництва;
- невиконання вимог технології, методів контролю та випробувань, системи якості, якщо перераховані зміни можуть викликати невідповідність продукції вимогам, контрольованим при сертифікації.

Рішення про зупинення дії сертифікату та ліцензії на право застосування знака відповідності не приймається в тому випадку, якщо шляхом коригувальних заходів, погоджених з органом з сертифікації, що його видав, заявник може усунути виявлені причини невідповідності та підтвердити без повторних випробувань в

акредитованій лабораторії, відповідність продукту або процесів нормативним документам. Якщо цього зробити не можна, то дія сертифіката відміняється, і ліцензія на право застосування знака відповідності анулюється. Інформація про призупинення або скасування дії сертифіката доводиться органом з сертифікації, що його видав, до відомості заявника, споживачів та інших зацікавлених організацій. Дія сертифікату та право маркування продукції знаком відповідності можуть бути відновлені при виконанні підприємству-розробнику наступних умов:

- виявлення причин невідповідності та їх усунення;
- подання до органу з сертифікації звіту про виконану роботу щодо поліпшення та забезпечення якості продукції;
- проведення за методиками і під контролем органу сертифікації додаткових випробувань продукції та отримання позитивних результатів.

2.3.2 Документування процесів і результатів сертифікації програмних продуктів

Склад і зміст документації для сертифікації системи якості підприємства залежать від характеристик проектування, розроблення та модифікації програмних засобів, а також від вимог до їх якості та особливостей технологічного середовища. Тому необхідний комплект документів для кожного підприємства чи проекту слід вибирати й адаптувати стосовно до цих характеристик. Оцінюваними при сертифікації показниками системи якості є наявність відповідних документів та практичне виконання вимог певного рівня моделі зрілості CMMI або адаптованого профілю стандартів на базі ISO 9000:2000, а також, створених на їх основі, посадових інструкцій фахівцями підприємству-розробнику. Заявник повинен підготувати і пред'явити випробувальній лабораторії узгоджений між замовником і розробником і затверджений комплект документів для перевірки їх достовірності, достатності складу і якості виготовлення відповідно до нормативних документів.

Орієнтовний комплект основних документів при сертифікації складається з трьох груп:

– базові нормативні документи систем якості відповідно до номенклатури і змістом профілю стандартів на базі ISO 9000:2000 або моделі зрілості CMMI, а також підготовлені розробниками на їх основі програма, керівництво та інструкції, які надаються випробувачам (експертам) системи якості або продукції, що перевіряється;

– вихідні документи, що характеризують конкретне підприємство або проект, а також життєвий цикл програмного засобу, підготовлювані керівництвом проекту для сертифікації його якості;

– звітні документи випробувачів, що відображають результати перевірки (сертифікації) системи якості підприємства та/або програмного продукту, що подаються органу сертифікації, заявнику і керівництву підприємства, що перевіряється.

Програмний продукт або система якості підприємства, що подаються на сертифікацію, повинні йти в комплекті з відповідною документацією. Перелік і приблизний зміст груп цих документів орієнтовані на загальний випадок перевірки систем якості підприємств, що забезпечують життєвий цикл великих програмних продуктів. Комплект документів може скорочуватися і адаптуватися за погодженням між заявником, сертифікатором і керівництвом підприємства, що перевіряється відповідно до характеристик проектів програмних засобів. Деякі документи можуть об'єднуватися в інтегровані звіти з чіткою відповідальністю певних фахівців за їх виконання.

Базові документи системи якості підприємства та життєвого циклу програмного засобу:

1. Концепція, термінологія, вимоги та настанови щодо поліпшення діяльності – системи менеджменту якості – *ISO 9000:2000* або версія моделі зрілості *CMMI*.

2. Адаптовані версії або перелік розділів і рекомендацій стандартів *ISO 12207*, *ISO 15504*, їх змін і посібників із застосування, виділених при адаптації та обов'язкових для використання в системі якості конкретного підприємства або проекту програмного продукту.

3. Адаптована версія або перелік розділів і рекомендацій стандарту *ISO 900003*, виділених при адаптації та обов'язкових для застосування в системі якості підприємства, що випускає програмний продукт.

4. Базові характеристики і атрибути якості проекту ПЗ, виділені, адаптовані і конкретизовані на основі стандартів *ISO 12182*, *ISO 9126*, *ISO 14598*, *ISO 25000*.

5. Адаптована версія і затверджена редакція керівництва по супроводу та конфігураційному управлінню основі рекомендацій стандартів *ISO 14764*, *ISO 10007*, *ISO 15846*.

6. Комплект посадових інструкцій, що визначають відповідальність, повноваження і порядок взаємодії всього керівного, виконує і перевіряючого роботу персоналу, який бере участь в процедурах системи якості підприємства для конкретного проекту ПЗ.

Вихідні документи, що відображають особливості життєвого циклу конкретного програмного засобу:

1. Опис характеристик програмних продуктів, що створюються на підприємстві, системи і зовнішнього середовища їх життєвого циклу, необхідних для адаптації та підготовки робочих версій стандартів і вимог проекту ПЗ та системи якості підприємства відповідно до рекомендацій стандартів *ISO 12207*, *ISO 15504*, *ISO 90003* та *ISO 9126*.

2. Опис цілей, вимог і зобов'язань підприємства-розробника в області системи якості, критеріїв якості процесів і продуктів розроблення, поставки та підтримки всього життєвого циклу ПЗ.

3. Комплект експлуатаційних документів, що поставляються замовнику і користувачам для забезпечення ЖЦ та застосування конкретної версії програмного продукту на основі адаптованих стандартів *ISO 9294*, *ISO 15910*, *ISO 18019*.

4. Документація і засоби автоматизації проектування, розроблення, модифікації, контролю та випробувань, що використовуються для забезпечення життєвого циклу програмного продукту.

5. Плани і методики випробувань застосування та оцінки ефективності процесів системи якості підприємства та програмного продукту.

6. Методики супроводу, ідентифікації компонентів програмного продукту та документації, аналізу та затвердження версій комплексів програм і даних.

7. Методика конфігураційного керування, затвердження, зберігання, захисту, копіювання версій програмного продукту та супроводжуваних документів, а також накопичення і зберігання,

zareєстрованих в архіві підприємства даних про характеристики якості протягом життєвого циклу версій програмного продукту.

Результуючі документи випробувань – сертифікації системи якості підприємства та/або програмного продукту:

1. Звіт про наявність, актуальності і систематичності оформлення документації, адаптованої до вимог і положень системи якості підприємства, що забезпечує інтегрований процес гарантії якості протягом усього життєвого циклу програмного продукту.

2. Результати контролю та випробувань стану і застосування системи якості, що проводяться періодично для визначення її придатності й ефективності.

3. Звіт про наявність та підтримці в робочому стані методик проведення перевірок і документально оформлених звітів про результати досягнутого якості виконання вимог договору на сертифікацію з замовником.

4. Результати реєстрації досягнутих характеристик якості комплексу програм: ідентифікація, накопичення, зберігання zareєстрованих даних про характеристики та атрибутах якості програмного продукту і його компонентів.

5. Результати реалізації плану розроблення, документально оформлених вхідних і вихідних даних етапів розроблення та протоколів перевірки реалізації життєвого циклу ПЗ.

6. Результати практичного виконання програми якості та здійснення регламентованої діяльності в сфері якості на всіх етапах життєвого циклу ПЗ.

7. Результати атестації імітаторів зовнішнього середовища і генераторів тестів, а також оцінка їх достатності для виконання сертифікаційних випробувань програмного продукту.

8. Результати аналізу виконання планів і методик проведення випробувань, протоколи випробувань, оцінки відповідності результатів випробувань пропонованим вимогам, а також результати випробувань, затверджені представниками заявника, замовника і постачальника.

9. Акт результатів перевірок реальних характеристик життєвого циклу ПЗ і системи якості підприємства, висновки про їх відповідність вимогам до сертифікації виробництва програмного продукту.

10. Сертифікат системи якості підприємства та / або програмного продукту і забезпечення його життєвого циклу, ліцензія на застосування знаків відповідності.

2.4 Практичні завдання

Завдання 1. Напишіть реферат на одну із тем:

- аналіз стандарту для систем керування якістю «*Bootstrap*»;
- аналіз стандарту для систем керування якістю «*Trillium*»;
- аналіз стандарту для систем керування якістю «*ISO 12207*».

Порівняйте розглянутий стандарт з *ISO 9001*, *CMM* та *SPICE*.

Завдання 2. Оберіть необхідний рівень зрілості організацій згідно моделі *CMM* для наступних проектів, що наведені у додатку В.

Завдання 3. Відповідно до наведеної моделі оцінювання розробіть програму для підрахунку рівня *CMM* для обраної системи.

Завдання 4. Розробіть необхідний комплекс документів для сертифікації проекту з додатку В:

- а) на відповідність *CMM*;
- б) на відповідність *ISO*;
- в) на відповідність ДСТУ ISO/IEC 15504.

2.5 Контрольні запитання

1. З яких процесів складається система сертифікації?
2. Які документи потрібні для створення органу сертифікації?
3. Які розділи містить керівництво з якості?
4. З яких етапів складається процес сертифікації програмних продуктів?
5. Які види робіт містить перевірка системи якості?
6. Які документи необхідно підготувати для сертифікації?
7. Які стандарти якості програмного забезпечення необхідно використовувати на різних етапах життєвого циклу програмного продукту?
8. В чом полягає мета процесу забезпечення якості?

9. Наведіть визначення системи якості.

10. Наведіть основні характеристики міжнародного стандарту *ISO 9001*.

11. Які характеристики повинні бути для отримання певного рівня зрілості відповідно до *CMM*?

12. Наведіть порівняльну характеристику стандартів *CMM* та *SPICE*.

2.6 Література до розділу

Сертифікація: [22-32],[36-39],[44],[46],[50],[52],[102],[101]; ISO 9001: [25],[115]; CMM: [66-69],[104],[106],[109-110]; PSP/TSP: [115]; ISO 15505:[82-90].

3 ВИТРАТИ НА ЯКІСТЬ

3.1 Вартість якості

Поняття ціни якості спочатку була введена Дж. Джураном (*J.M Juran*) і Ф. Гріном (*F.M Gryna*) як вартість в складі продукту, що може бути збережена, якщо всі виконавці працюють бездоганно.

Ціна якості – важлива категорія, оскільки фактично вона відображає вартість робіт на доопрацювання, збільшену вартість супроводу.

Ціна якості може бути розділена на два головних типи: узгоджена (*conformance*) і неузгоджена (*non-conformance*).

Узгоджена ціна якості – це сума, витрачена на досягнення якості продукту. Вона ділиться на ціну попередження (*Prevention cost*) і ціну контролю (*Appraisal cost*). Витрати попередження пов'язані з попередженням дефектів перш ніж вони відбудуться. При розробці ПЗ прикладом витрат на попередження є витрати на навчання колективу базовим методологіям, перехід на сучасні технології розробки, використання автоматизованих засобів проектування і розроблення.

Ціна контролю містить вимірювання, оцінювання або ревізії продукту для забезпечення гарантії відповідності між вимогами, стандартами якості і результатом. Для розробки програмного забезпечення, наприклад, витрати на оцінювання можуть містити інспекцію програм, тестування та вимірювання програмних показників.

Неузгоджена ціна якості складається з усіх витрат понесених внаслідок виявлення недоліків, виникнення помилок і виходу з ладу.

Внутрішні витрати пов'язані з проблемами, виявленими до відправлення продукту замовнику. Для розробки програмного забезпечення до них можуть бути віднесені витрати на переробку програм, повторну інспекцію та тестування. Витрати пов'язані з помилками, що виявилися при експлуатації продукту у замовника, відносяться до зовнішніх витрат. Для програмного забезпечення, наприклад, вони містять витрати на супровід і підтримку, збитки від простоїв і некоректного функціонування.

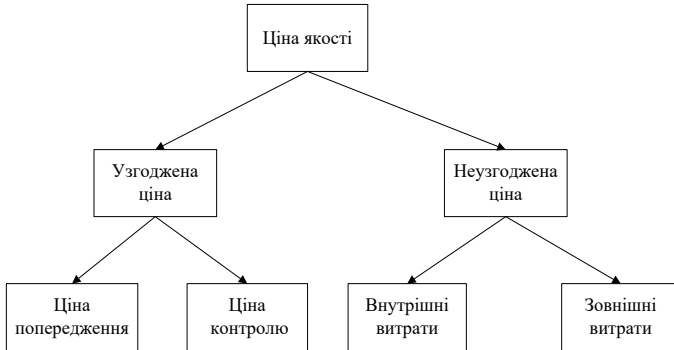


Рис. 3.1. Ієрархія поняття «ціна якості»

Статистичні дослідження впливу процесів розробки на якість результуючого продукту показують, що:

1. Удосконалення процесу розробки та впровадження програмного забезпечення значно зменшують відносну (у перерахунку на одиницю об'єму програмного продукту в обраній метриці) неузгоджену вартість якості при збереженні узгодженої вартості якості на колишньому рівні.

2. Інвестиції у вдосконалення процесу розробки програмного продукту за умови раннього і адресного впровадження процедур поліпшення якості ведуть до значного скорочення дефектів і дають високий позитивний економічний ефект (рис. 3.2).

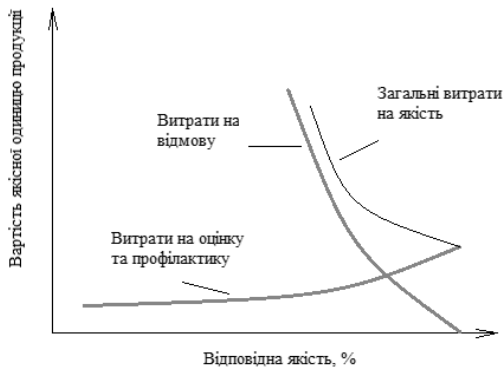


Рис. 3.2. Ціна якості

3.2 Функції втрат

Вибір і формування вимог до ПЗ полягає в аналізі необхідних властивостей, що характеризують якість їх функціонування з урахуванням технологічних і ресурсних можливостей розробників. При цьому під якістю функціонування будемо розуміти безліч властивостей, що обумовлюють придатність ПЗ забезпечувати надійне і своєчасне подання необхідної інформації споживачу для її подальшого використання за призначенням. Відповідно до принципових особливостей, призначення та властивостей кожного ПЗ при проектуванні повинні вибиратися номенклатура і значення характеристик якості, необхідних для ефективного застосування користувачами, які згодом відбиваються в специфікаціях вимог і у технічній документації до кінцевого продукту.

Кожна характеристика якості може використовуватися, якщо визначена її метрика, міра і шкала і може бути вказаний спосіб її виміру та зіставлення з потрібним значенням. Для конкретних ПЗ домінуючі критерії якості виділяються при проектуванні та визначаються вимогами технічного завдання і функціональним призначенням. Вони повинні, перш за все, відображати функціональну придатність для застосування із заданими цілями.

При розрахунку внутрішніх та зовнішніх втрат можливо використовувати класичну функцію або функцію втрат Г. Тагуті.

Класична або дискретна функція втрат (3.1), встановлює чіткий допуск для кожної характеристики, та якщо показник в межах вказаного допуску то втрати вважаються нульові (рис.3.3), і якщо показник вийшов за межі то втрати вважаються постійними.

$$L(y) = \begin{cases} 0, & y_a \leq y \leq y_b \\ A_1, & y < y_a \\ A_2, & y > y_b \end{cases} \quad (3.1)$$

де A_1, A_2 – втрати при виході за ліву та праву межі допуску.

До переваг класичної функції втрат можна віднести наступні:

- 1) дозволяє чітко надати формулювання придатності та браку;
- 2) добре розроблений математичний для проектування засобів контролю точності виготовлення.

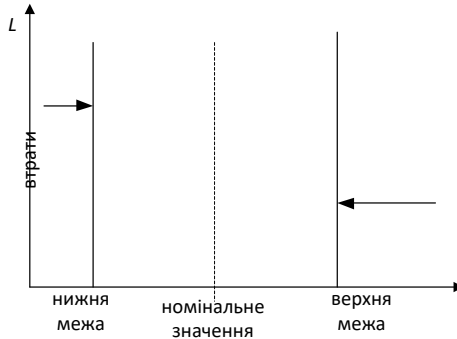


Рис. 3.3. Класична функція втрат

Недоліки:

- відбувається розмиття поняття «номінальне значення», тобто замість нього оперують поняттями значень полів допусків;
- показники, що значення яких знаходяться у межах допуску вважаються однаково вірними;
- такий підхід не дозволяє враховувати характер похибки.

Альтернативною функцією втрат є функція втрат Г.Тагуті (рис. 3.4):

$$L(y) = k \cdot (y - \tau)^2, \quad (3.2)$$

де τ - номінальне значення, y – значення показника.

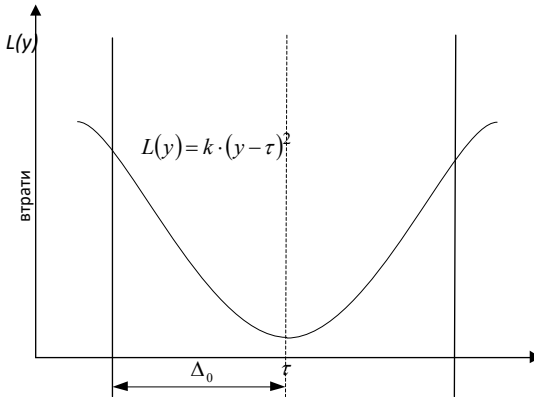


Рис. 3.4. Функція втрат Г. Тагуті

На відміну від класичної дискретної функції втрат дана функція неперервна, причому із збільшенням варіації втрати зростають. У міру зменшення відхилень від номінального значення втрати зменшуються і стають рівними нулю в точці, що відповідає номінальному значенню

Переваги використання квадратичної функції втрат:

- конструювання рейтингу значущості показників якості;
- гнучкий підхід до встановлення граничних значень для показників.

3.3 Технології оцінювання трудовитрат

Нижче даються описи ряду поширених технологій оцінювання трудовитрат в порядку зростання їх складності.

1 Облік обмежень, що накладаються фінансовим кошторисом або графіком виконання робіт.

2 Аналогія з попередніми проектами.

3.Експертна оцінка. Кількість виконавців або терміни, які необхідних для виконання потрібних завдань, підраховують один або більше число експертів. Цей метод може виглядати дуже простим, коли експерт записує свою оцінку на шматку паперу, або ж дуже складним, коли потрібний консенсус всіх приймаючих в ній участь (прикладом може слугувати технологія Wideband Delphi).

Методи декомпозиції. Якщо програмний продукт досить великий та складний, по всій ймовірності, отримання оцінок трудовитрат на розробку і тестування цього продукту, вимагатиме великих витрат часу і зусиль. Цілком імовірно, що в даній ситуації буде призначений керівник проекту, відповідальний за дотримання фінансового кошторису і графіку виконання робіт всього проекту, а від групи тестування буде потрібно надання керівнику проекту вихідних даних в спеціальному форматі. Керівник проекту може доручити кожній групі фахівців дати власну оцінку витрат із застосуванням однієї з описаних вище технологій. З іншого боку, керівник може скористатися більш уніфікованим підходом, в рамках якого програмний продукт розбивається на блоки або за кількістю рядків коду, або за функціональними балами, а потім до блоків застосовується деякий оцінюючий алгоритм. Якщо використовується саме такий підхід, то групі тестування доцільно отримати власну незалежну оцінку, скажімо, експертну, та переконатися в тому, що застосування алгоритмічного підходу має сенс.

Моделі емпіричного оцінювання. Існує безліч моделей оцінювання, які можуть використовуватися для обчислення витрат на розробку проекту зі створення програмного продукту. В основу цих моделей покладено кількість рядків програмного коду (LOC – number of lines) або функціональні бали (FP – functional points), причому для одних и тих же вихідних даних ці моделі дають різні результати. Ключовою умовою впровадження будь-якої з моделей є калібрування моделі відносно локальних умов за рахунок її застосування на завершених проектах та налаштування її на фактичні дані так, щоб вона видавала передбачувані результати.

Після того, як завдання визначені, а трудовитрати на кожну задачу підраховані, потрібно визначити час вирішення кожного завдання та дослідити перший розріз графіка робіт.

Як тільки буде отримана оцінка кількості виконавців, обладнання та часу, необхідного для тестування програмного продукту, потрібно оцінити ризики, які пов'язані з цими оцінками. Це робиться виходячи з припущення, що може вийти з ладу під час проведення випробування. Далі нескладно скласти план усунення виникаючих проблем. Обсяг трудовитрат, необхідних для пом'якшення наслідків від уречливих ризиків, дає уявлення про те, наскільки отримані оцінки можуть розрізнятися з фактичними

значеннями витрат часу і коштів. Ризики, вбудовані в отримані оцінки, — це перше, про що слід доповісти вищому керівництву в процесі перегляду оцінок.

Формула Боема для розрахунку трудовитрат на доробку полягає в наступному:

$$AME = ACT \times SDT, \quad (3.3)$$

де AME — є річними трудовитратами на підтримку, вимірюваними в людино-місяцях; ACT — є річним трафіком змін, що відображає частину вимог вихідного продукту, які пройшли деякі зміни під час типового року за допомогою додавання або модифікації; SDT — це час розроблення ПЗ в людино-місяцях.

Припустимо, проект з розроблення ПЗ вимагав 100 людино-місяців зусиль, і було прогнозовано, що 15% всього коду буде змінено за рік. Основна оцінка річних зусиль підтримки (AME) буде дорівнювати:

$$AME = 0.15 \times 100 = 15 \text{ людино-місяців.}$$

Іншими словами, варто спланувати витрати в розмірі 15 людино-місяців на рік на підтримку даного проекту з розроблення ПЗ.

Основний прогноз витрат річної підтримки може бути переглянутий шляхом оцінювання важливості кожного фактора, що впливає на витрати та вибір відповідного множника, прогножуючого відсоток зміни коду. Основні витрати на підтримку потім будуть помножені на кожен множник витрат для отримання переглянутого прогнозу витрат на підтримку.

Припустимо, у минулій системі факторами, які мали найбільший вплив на витрати підтримки, були складність продукту ($CPLX$), яка дуже висока, і доступність підтримки з боку співробітників ($AEXP$), яка дуже низька.

Якщо $CPLX = 1.30$ и $AEXP = 1.29$, тоді:

$$AEM = 15 \times 1.30 \times 1.29 = 25.2 \text{ людино-місяців.}$$

3.4 Практичні завдання

Завдання 1. Оберіть один з проектів (додаток В).

1.1 Проаналізуйте та підготуйте специфікацію на обраний проект.

1.2 Визначте потреби у матеріальних та трудових ресурсах.

1.3 Розрахуйте та оцініть витрати на розробку програмного продукту методами функціональних точок та СОСОМО II.

Завдання 2. Визначте чинники, на які впливає автоматизована обробка даних та економічно оцініть їх зміни.

Завдання 3. Розрахуйте втрати від виходу за терміни розробки з використанням класичної функції втрат та функції втрат Г.Тагута. Порівняйте результати.

3.5 Контрольні запитання

1. Поясніть поняття «ціна якості», розкрийте різницю між узгодженою та неузгодженою ціною якості.

2. Як впливає якість процесів розробки на якість результуючого продукту?

3. Розпишіть загальний алгоритм оцінювання розроблення системи.

4. Опишіть методи *FPA IFPUG* та *COCOMO II*, поясніть відмінності між ними.

5. Розкрийте поняття «функціональні точки»?

6. Які вам відомі технології оцінювання трудовитрат?

7. Чим відрізняються класична функція втрат та функція втрат Г.Тагута?

3.6 Література до розділу

Методи оцінки вартості: [Ошибка! Источник ссылки не найден.], [9], [59-60], [65], [71-72].

ЧАСТИНА 2 МЕТРИКИ ЯКОСТІ ІНФОРМАЦІЙНИХ СИСТЕМ

4 МОДЕЛІ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Модель якості програмної системи це множина взаємопов'язаних характеристик, що утворюють базис для специфікації вимог до якості та оцінювання якості.

4.1 Модель якості за МакКолом

Першою широко відомою моделлю якості ПЗ стала модель, запропонована у 1977 МакКолом і іншими.

У ній характеристики якості розділені на три групи:

1. **Фактори** (*factors*), що описують ПЗ з позицій користувача та визначаються вимогами;
2. **Критерії** (*criteria*), що описують ПЗ з позицій розробника та задаються як цілі;
3. **Метрики** (*metrics*), що використовують як кількісний вимір якості.

Фактори якості, об'єднуються в три групи в залежності від способів роботи людей з ПЗ. Всього було визначено 11 факторів. Отримана структура зображена у вигляді трикутника МакКола (рис. 4.1).

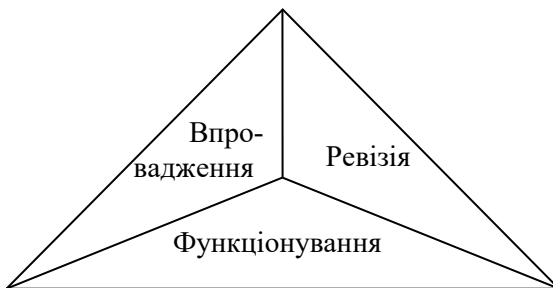


Рис. 4.1. Трикутник МакКола

Критерії якості – це числові рівні факторів, що сформульовані як цілі при розробці.

Об'єктивно оцінити або виміряти фактори якості безпосередньо досить важко. Тому, МакКол ввів метрики якості, які з його точки зору легко вимірювати й оцінювати. Оцінки в його шкалі набувають значень від 0 до 10. Були запропоновані такі метрики:

- зручність перевірки на відповідність стандартам (*auditability*);
- точність керування і обчислень (*accuracy*);
- ступінь стандартності інтерфейсів (*communication commonality*);
- функціональна повнота (*completeness*);
- однорідність використовуваних правил проектування та документації (*consistency*);
- ступінь стандартності форматів даних (*data commonality*);
- стійкість до помилок (*error tolerance*);
- ефективність роботи (*execution efficiency*);
- розширюваність (*expandability*);
- широта області потенційного використання (*generality*);
- незалежність від апаратної платформи (*hardware independence*);
- повнота протоколювання помилок та інших подій (*instrumentation*);
- модульність (*modularity*);
- зручність роботи (*operability*);
- захищеність (*security*);
- самодокументованість (*selfdocumentation*);
- простота роботи (*simplicity*);
- незалежність від програмної платформи (*software system independence*);
- можливість простежування вимог (*traceability*);
- зручність навчання (*training*).

Кожна метрика впливає на оцінку кількох факторів якості. Числовий вираз фактора являє собою лінійну комбінацію значень метрик, що впливають на нього. Коефіцієнти цього виразу

визначаються по-різному для різних організацій, команд розроблення, видів ПЗ, використовуваних процесів і т.п.

4.2 Модель якості за Боемом

У 1978 Боем запропонував свою модель, що по суті є розширенням моделі МакКола.

Атрибути якості поділяються за способом використання ПЗ (*primary use*).

Визначено 19 проміжних атрибутів (*intermediate construct*), які включають всі 11 факторів якості за МакКолом. Проміжні атрибути поділяються на примітивні (*primitive construct*), які, в свою чергу, можуть бути оцінені на основі метрик.

До факторів МакКола Боем додає наступні: ясність (*clarity*), зручність внесення змін (*modifiability*), документованість (*documentation*), здатність до відновлення функцій (*resilience*), зрозумілість (*understandability*), адекватність (*validity*), функціональність (*functionality*), універсальність (*generality*), економічна ефективність (*economy*).

Модель Боєма широко використовується для прогнозування витрат на підтримку певного рівня якості. Її відносно легко зрозуміти, і, що більш важливо, вона дозволяє переглянути прогнози завдяки безлічі показників витрат.

4.3 Модель якості FURPS

В 1987 г. Компанія *Hewlett Packard* запропонувала модель якості *FURPS* (*Functionality, Usability, Reliability, Performance, Supportability*) – індустріальну інтерпретацію засновану на моделях МакКола и Боєма. Модель містить п'ять високорівневих атрибутів, відповідно до назви: функціональність, зручність використання, надійність, продуктивність, супроводженість, що деталізуються 27 атрибутами низького рівня.

Сучасна модель *FURPS+* містить додаткові атрибути: вимоги до проекту (*Design requirements*), вимоги до реалізації (*Implementation requirements*), вимоги до інтерфейсу (*Interface requirements*), вимоги до фізичних компонент (*Physical requirements*).

Створення аналітичної документації згідно з *FURPS+* дозволяє створити якісно задокументований проект.

До недоліків можна віднести:

1. Хоча модель *FURPS* застосовна до всіх видів програмного забезпечення, її можна доповнити атрибутами якості, що відносяться до веб-додаткам: знаходжуваність, доступність і правову відповідність;

2. За структурою ця модель схожа на моделі МакКола и Боема, але вона не приділяє достатньо уваги до переносимості ПЗ.

4.4 Модель якості ISO 9126

Основою формального регламентування показників якості ПЗ є міжнародний стандарт *ISO 9126:1991* (ГОСТ Р ИСО / МЭК 9126-93) – "Програмна інженерія. Оцінювання програмного продукту. Характеристики якості та настанови щодо їх застосування". Розвиток цього міжнародного стандарту проводиться в напрямку уточнення, деталізації та розширення описів характеристик якості комплексів програм. Для заміни редакції 1991 року створено стандарт *ISO 9126-1-4*, що складається з чотирьох частин:

Частина 1: Модель якості – *ISO/IEC 9126-1:2001 Software engineering – Product quality – Part 1: Quality model*;

Частина 2: Зовнішні метрики якості – *ISO/IEC TR 9126-2:2003 Software engineering – Product quality – Part 2: External metrics* (ДСТУ *ISO/IEC TR 9126-2:2008* Програмна інженерія. Якість продукту. Частина 2. Зовнішні метрики);

Частина 3: Внутрішні метрики якості – *ISO/IEC TR 9126-3:2003 Software engineering – Product quality – Part 3: Internal metrics*;

Частина 4: Метрики якості у використанні – *ISO/IEC TR 9126-4:2004 Software engineering – Product quality – Part 4: Quality in use metrics*.

У першій частині стандарту *ISO 9126-1* наводиться схема взаємозв'язку частин стандарту *ISO 9126* і частин стандарту *ISO 14598*, а також область застосування, нормативні посилання, терміни та визначення. Визначається модель характеристик якості ПЗ та її зв'язок з життєвим циклом. Модель деталізується в наступних частинах стандарту. Вимоги користувача до якості в специфікаціях повинні в процесі верифікації перетворюватися у вимоги до зовнішньої якості, а потім у вимоги до внутрішньої якості. Процеси реалізації вимог до внутрішньої якості повинні забезпечувати зовнішню якість, а останнє – втілюватися в якість для користувачів. Коротко описані компоненти перетворення і реалізації вимог до складових якості цієї моделі.

Модель внутрішніх і зовнішніх характеристик якості ПЗ складається з шости груп базових показників, кожна з яких деталізована кількома нормативними субхарактеристиками:

Функціональна придатність деталізується:

- придатністю до застосування;
- коректністю (правильністю, точністю);
- здатністю до взаємодії;
- захищеністю.

Надійність рекомендується характеризувати:

- рівнем завершеності (відсутності помилок);
- стійкістю до дефектів;
- відновлюваністю;
- доступністю – готовністю.

Ефективність рекомендується відображати:

- тимчасовою ефективністю;
- використовуваністю ресурсів.

Застосовність (практичність) пропонується описувати:

- зрозумілістю;
- простотою використання;
- вивчаємістю;
- привабливістю.

Супровідність пропонується представляти:

- зручністю для аналізу;
- змінністю;
- стабільністю;

- тестованістю.

Переносимість (мобільність) пропонується відображати:

- адаптованістю;
- простотою установки – інсталяції;
- співіснуванням – відповідністю;
- заміщенням.

Додатково кожна характеристика супроводжується субхарактеристикою узгодженість, що повинна відображати відсутність протиріч з іншими стандартами та нормативними документами, а також з іншими показниками в даному стандарті. У стандарті *ISO 9126* відсутні методики кількісного вимірювання характеристик і порівняння з вимогами специфікацій, а також рекомендації, на яких етапах ЖЦ ПЗ їх доцільно застосовувати.

У стандарті виділена модель характеристик якості у використанні. У цій моделі використовуються інші базові характеристики в порівнянні з моделлю внутрішньої і зовнішньої якості. Рекомендуються такі основні характеристики якості ПЗ у використанні:

- системна ефективність застосування програмного продукту за призначенням;

- продуктивність – продуктивність при вирішенні основних завдань ПЗ, що досягається при реально обмежених ресурсах в конкретному зовнішньому середовищі застосування;

- безпека – надійність функціонування комплексу програм і можливий ризик від його застосування для людей, бізнесу і зовнішнього середовища;

- задоволення вимог і витрат користувачів відповідно до цілей застосування ПЗ.

Друга і третя частини стандарту *ISO 9126:2,3* присвячені формалізації відповідно зовнішніх і внутрішніх метрик характеристик якості складних ПЗ.

Взаємозв'язок метрик якості в цих частинах стандарту відображений однаковими моделями, аналогічними моделі першої частини стандарту. У стандарті внутрішня і зовнішня якості відносяться безпосередньо до самого програмного продукту, а метрики якості у використанні проявляються в ефекті від його застосування і залежать від зовнішнього середовища. Між трьома

типами метрик і характеристиками якості існує взаємовплив зверху вниз і залежність від низу до верху.

Викладено зміст і загальні рекомендації щодо використання відповідних метрик і взаємозв'язків між типами метрик. Основний зміст метрик якості в цих частинах стандарту описано у восьмому розділі у вигляді описів та 27 ілюстративних таблиць до них. Для цього в попередньому сьомому розділі описані рубрики та рекомендації, як читати і використовувати таблиці метрик субхарактеристик і атрибутів якості ПЗ.

Четверта частина стандарту *ISO 9126-4* призначена для покупців, постачальників, розробників, супроводжуючих, користувачів і менеджерів якості ПЗ. У ній повторена концепція трьох типів метрик, а також анотовані рекомендовані види вимірювань характеристик ПЗ: прямі, непрямі та індикатори властивостей (категорійні). Розглянута модель якості у використанні. Відзначаються необхідність ідентифікації призначення і специфіки споживачів програмного продукту, особливості вибору цілей оцінювання якості для різних сфер і етапів застосування ПЗ. Обґрунтовуються і коментуються виділені показники сфери (контексту) використання ПЗ та групи вибраних метрик для користувачів. На відміну від характеристик, описаних у попередніх частинах стандарту, в цій частині для якості у використанні рекомендується чотири: ефективність; продуктивність; задоволення вимог і захищеність.

Наводяться докладні рекомендації для опису та оцінювання виділених характеристик якості у використанні: специфіка визначення цілей і контексту середовища користувача; вибір, селекція і інтерпретація кожної з виділених метрик; виділення та затвердження критеріїв для реалізації та оцінювання якості; інтерпретація результатів вимірювань. Надано рекомендації з проектування процесів оцінювання, з оформлення тестів користувача, узагальнення зібраних даних і підготовки звітів про результати оцінювання якості програмного продукту.

4.5 Практичні завдання

1. Ви замовник програмного забезпечення. Створіть перелік вимог до фірми розробника та план опиту фахівця з якості з метою визначити фактори якості на даній фірмі.

2. Оберіть метрики оцінки якості, які ви вважаєте критичними. Обґрунтуйте.

3. Оберіть модель якості ПЗ, яка, на вашу думку, нині найбільш актуальна для України (моделі МакКола, Боема, *FURPS*, *ISO 9126*).

4.6 Контрольні запитання

1. Наведіть визначення моделі якості програмної системи.

2. На які групи розділені характеристики якості в моделі МакКола?

3. Які метрики були запропоновані МакКолом?

4. Які фактори додав Боем до моделі МакКола?

5. Які міжнародні стандарти були прийняті як держстандарт України для визначення внутрішніх та зовнішніх метрик якості?

4.7 Література до розділу

Моделі якості: [Ошибка! Источник ссылки не найден.],[7-8],[23],[33-34],[41],[50],[79-81], [101].

5 МЕТРИКА ЯК ОСНОВА ВИМІРЮВАННЯ

5.1 Метрика як основа виміру

Атрибути програмної системи, що характеризують її якість, вимірюються з використанням метрик якості.

Метрика – це комбінація конкретного методу вимірювання (способу отримання значень) атрибута сутності та шкали вимірювання (засобу, використовуваного для структурування одержуваних значень).

Метрика визначає *міру атрибуту* – змінну, якій присвоюється значення в результаті вимірювання. Наприклад, сутність – «звіт про виявлені дефекти», атрибут – «список дефектів», метод вимірювання – «підрахунок кількості дефектів у списку», шкала – «цілі числа більше нуля», міра атрибуту – «загальне число дефектів», ім'я метрики (зазвичай однойменне мірою) – «загальне число дефектів».

Міра атрибута може бути безпосередньою, якщо вона не залежить від заходів інших атрибутів, або непрямою, отриманої по заходам інших атрибутів.

За визначенням стандарту *ISO/IEC 9126-2* метрика якості програмної системи являє собою «модель вимірювання атрибута, який пов'язує з характеристикою якості ПЗ. Служить індикатором одного або багатьох атрибутів.

Метрика називається *базовою*, якщо в її основі лежить елементарний метод (примітив) вимірювання атрибута.

Для того щоб правильно користуватися результатами вимірювань, для кожної міри потрібно ідентифікувати шкалу вимірювання.

Стандарт *ISO/IEC 9126-2* рекомендує застосовувати 5 видів шкали вимірювання значень (впорядкованих від менш суворих до більш суворим):

– *номінальна шкала*. Це класифікаційна шкала, що виконує категоризацію властивостей об'єкта, що оцінюється. Категорії не впорядковані. Наприклад, дефекти можуть класифікуватися на дефекти інтерфейсу, логіки, оголошення даних та ін.;

– **порядкова шкала.** Дозволяє упорядковувати характеристики за зростанням або убуттям шляхом порівняння їх з базовими значеннями. Наприклад, для рівня серйозності наслідків події шкала може містити значення «низький», «середній», «високий», «критичний». Для рівнів СММ – {1, 2, 3, 4, 5}. Відстань між значеннями за шкалою не грає ролі. Характеристики, що мають номінальну або порядкову шкалу вимірювання, називаються якісними.

– **інтервальна шкала.** Відзначає істотні відмінності властивостей об'єкта, «дистанцію» між ними (наприклад, календарні дати або значення щільності дефектів. Використовується в арифметичних операціях і операціях порівняння. Нульове значення не припустимо;

– **відносна шкала.** Значення за цією шкалою розрізняються по відношенню до вибраної одиниці (наприклад, часу, змінюваному від 0 до безкінечності, або вартості). Застосовуючи цю шкалу можна розрахувати, наприклад, час між відмовами, розмір програмного компонента в SLOC та ін. Вважається найбільш кращою шкалою вимірів. Дозволяє застосовувати широкий спектр інструментів вимірювання (гістограм, діаграм Парето та ін.);

– **абсолютна шкала.** Це спеціальний випадок відносної шкали. У цій шкалі вказується абсолютне значення величини. Наприклад: «розмір програми дорівнює 2К», «число виявлених помилок одно 20».

Вимірює значення метрики саме по собі не несе інформації про рівень задоволення вимог до якості. Для цих цілей шкала повинна бути розділена на області (ранги), які відповідають різним ступеням задоволення вимог. Приклади розподілу шкали:

– розподіл значень за двома категоріями – задовільні і незадовільні значення;

– поділ шкали за чотирма категоріями, обмеженим трьома рівнями значень – поточним, гіршим і плановим.

По мірі накопичення практики вимірювань і знань про вивчаємі атрибути шкал їх вимірювання можуть еволюціонувати від менш інформативних (номінальної і порядкової) до більш інформативним (відносної чи абсолютної).

5.1.2. Класифікація мір якості

Для розробки процедур збирання даних, інтерпретації заходів та їх нормалізації з метою порівняння, потрібно розрізняти такі типи міри, визначених метриками.

Міри розміру. Показують розмір ПЗ в різних одиницях виміру, наприклад:

- **функціональний розмір** – облік функціональних можливостей ПЗ;
- **розмір програми** – облік кількості рядків вихідного коду, кількості модулів, кількості операторів на мові програмування;
- **обсяг ресурсів**, що використовуються працюючою програмою – облік об'єму оперативної пам'яті, дискової пам'яті або мережевого трафіку, завантаженості процесора – кількості оброблюваних інструкцій у секунду і ін.;

Міри часу. Наводять періоди реального часу (в секундах, хвилинах або годинах), процесорного часу (в секундах, хвилинах або годинах роботи процесора) або календарного часу (в робочих годинах, календарних днях, місяцях, роках), наприклад:

- **час функціонування системи** з безперервною або дискретною роботою компонентів програмного забезпечення (облік минулого часу (при роботі програмного забезпечення в системі протягом встановленого періоду часу), облік часу з моменту включення (при безперервному використанні вбудованого програмного забезпечення системи реального часу), облік нормалізованого часу (при спільній роботі кількох комп'ютерів, що обмінюються даними);
- **час виконання** (час, необхідний працюючому програмному компоненту системи для завершення розв'язання певної задачі в певних умовах);
- **час використання** (час, що витрачається певним користувачем на вирішення завдань з допомогою ПЗ), наприклад: час сеансу роботи (від початку до завершення), час вирішення задачі і ін.;

Міри зусиль. Наводять корисний (продуктивний) час, пов'язаний з певним завданням проекту, наприклад:

- продуктивність праці (при індивідуальній роботі);

- трудомісткість (при колективній роботі);

Міри інтервалів між подіями. Це інтервали часу між настанням подій, що відбуваються в певний період спостереження, наприклад, час між послідовними відмовами. Замість цієї міри може використовуватися частота настання подій;

Лічильні міри. Являють собою статичні лічильники (для обліку певних елементів у робочих продуктах ПЗ (документах)) або кінетичні (динамічні) лічильники (для обліку подій або дій людини), наприклад:

- кількість виявлених помилок (облік помилок в ході інспекції, тестування, функціонування або супроводу ПЗ);

- структурна складність програми (кількість шляхів у програмі, цикломатична складність та ін.);

- кількість несумісних елементів (облік помилок узгодження (вимог, стандартів, форматів та ін.), облік відповідей типу «так» / «ні» у опитувальних);

- кількість змін – облік елементів конфігурації, в яких відбулися зміни;

- кількість виявлених відмов – облік відмов при тестуванні, функціонуванні чи супроводі ПЗ;

- кількість спроб – облік спроб корегування дефектів або помилок;

- ергономічні лічильники – число натиснень клавіш, клацань на кнопках та ін.

- лічильники-оцінки – облік результатів, зведених в опитувальних листах, контрольних листах та ін.)

При виконанні вимірювань базові міри розміру, часу і розхунки можуть використовуватися в різних комбінаціях. Вони служать основою для нормалізації та забезпечують можливість зіставлення метрик.

5.1.3 Класифікація метрик якості

Для зручності застосування загальних прийомів вимірювань метрики зазвичай класифікують як:

- **об'єктивні/суб'єктивні.** Об'єктивні метрики містять підрахунки елементів, які можуть бути незалежно перевірені

(кількість рядків коду, кількість помилок, складність та ін.) Вони знижують вплив особистої думки на обчислення й аналіз метрик. Суб'єктивні метрики ґрунтуються на індивідуальному чи колективному розумінні або перевазі певних характеристик або умов (рівень складності проблем, вартісні коефіцієнти та ін.);

- **примітивні/обчислювані.** Примітивні (базові) метрики можна безпосередньо спостерігати (розмір програми в KSLOC, кількість дефектів, знайдених при тестуванні та ін.) Обчислювані метрики не можуть безпосередньо спостерігатися, але можуть обчислюватися по примітивним метрик (число дефектів, що припадають на SLOC, трудомісткість та ін.);

- **динамічні/статичні.** Динамічним метрикам властивий компонент часу. Значення змінюються з плином часу, починаючи з моменту збирання даних (наприклад, кількість помилок в місяць). Статичні метрики інваріантні до часу (кількість виявлених дефектів, загальна трудомісткість робіт тощо);

- **прогнозуючі/пояснюючі.** Значення прогнозуючих метрик можуть бути отримані заздалегідь, наприклад, оцінювана інтенсивність відмов. Значення пояснюючих метрик з'являються пост-фактум, наприклад, реальна інтенсивність відмов.

По відношенню до виду об'єкта вимірювання (працююча програма або сукупність документів) міри та відповідні метрики підрозділяються на зовнішні, внутрішні і метрики використання ПЗ.

Зовнішні метрики використовують міри працюючого на комп'ютері програмного продукту, отримані в результаті вимірювання його поведінки в ході тестування і функціонування.

Зовнішні метрики розробляються з метою:

- демонстрації якості програмного продукту, за допомогою характеристик і підхарактеристик якості, на стадії тестування і експлуатації;

- використання для підтвердження (валідації) того, що програмний продукт задовольняє зовнішнім вимогам до якості;

- передбачення реальної експлуатаційної якості;

- визначення ступеня, в якій програмний продукт буде задовольняти встановленим і передбачуваним вимогам користувачів у ході реальної експлуатації.

Можна сказати, що сукупність зовнішніх метрик призначена для оцінювання зовнішньої якості – міри, в якій продукт задовольняє встановленим (заявленим) і що маються на увазі потребам при використанні в певних умовах.

Розробка зовнішніх метрик ґрунтується на виконанні наступних вимірювань:

- поведінки програмного продукту при тестуванні та функціонуванні в поєднанні з іншими програмними продуктами, апаратним забезпеченням або системою обробки інформації в цілому;
- поведінки користувача (сценаріїв використання ПЗ).

Під **виміром (оцінкою) поведінки** розуміється оцінка масштабів можливих наслідків неадекватної поведінки, які загрожують життю або здоров'ю людей, природним ресурсам, можуть призвести до руйнування даних, неузгодженості або недостовірності інформації, втрати безпеки, деградації сервісу (послуг), економічних втрат та ін.

Прикладами зовнішніх метрик для такої характеристики якості як надійність, можуть бути середній час між відмовами, кількість усунених дефектів при тестуванні, інтенсивність відмов та ін.

Внутрішні метрики забезпечують можливість користувачам, розробникам, тестувальникам і менеджерам оцінювати якість проміжних і кінцевих продуктів ПЗ безпосередньо за їх властивостями, без виконання на комп'ютері.

Внутрішні метрики розробляються таким чином, щоб вони могли:

- відображати якість проміжних і кінцевих програмних продуктів по тим характеристикам і підхарактеристикам якості, які визначені в моделі якості ПЗ;
- служити посібником до дії при плануванні та поліпшенні процесів, які впливають на проміжні та кінцеві продукти;
- використовуватися при верифікації того, що проміжні і кінцеві продукти задовольняють вимогам до внутрішньої якості ПЗ, передбаченим планами вдосконалення процесів;
- передбачати зовнішні метрики якості.

Можна сказати, що сукупність внутрішніх метрик призначена для оцінювання внутрішньої якості – безлічі атрибутів продукту, що визначає його здатність задовольняти встановленим або реальним потребам при використанні в певних умовах.

Розробка внутрішніх метрик ґрунтується на виконанні вимірювань статичних атрибутів, які визначені і можуть бути оцінені по тексту вихідного коду, графічному або табличному поданням потоків управління і даних, структур переходу станів або за документами ПЗ.

Прикладами внутрішніх метрик для надійності можуть бути кількість помилок, знайдених при інспекції коду, кількість усунених дефектів в результаті інспекції коду, прогнозована кількість помилок, що залишилися, та ін.

Метрики якості у використанні (метрики експлуатаційної якості) вимірюють ступінь, в якій програмний продукт, встановлений і експлуатований в певному середовищі, задовольняє потребам користувачів в ефективному, продуктивному і безпечному рішенні задач.

Метрики якості у використанні допомагають оцінити не властивості самої ПЗ, а видимі результати її експлуатації – експлуатаційну якість.

Очевидно, що для правильного вимірювання експлуатаційної якості важливо враховувати контекст застосування ПЗ – особливості категорій її користувачів, специфіку розв'язуваних ними завдань, а також фізичні та соціальні фактори середовища їх роботи.

Прикладами експлуатаційних метрик якості можуть бути точність і повнота досягнення цілей користувачів, продуктивність праці, ресурси, витрачені у зв'язку з досягненням ефективного вирішення завдань, думку користувачів щодо властивостей ПЗ та ін.

Внутрішні, зовнішні та експлуатаційні метрики якості взаємопов'язані. Досягнення експлуатаційної якості залежить від задоволення критеріїв зовнішньої якості, заснованих на зовнішніх мірах і метриках якості, які, в свою чергу, залежать від задоволення відповідних критеріїв внутрішньої якості, заснованих на внутрішніх мірах та метриках, пов'язаних із зовнішніми.

Зазвичай вимоги користувачів до якості специфікуються за допомогою зовнішніх метрик і експлуатаційних метрик якості, а внутрішні метрики вибираються таким чином, щоб вони могли використовуватися для передбачення значень зовнішніх метрик.

Побудувати строго теоретичну модель, що встановлює взаємозв'язок зовнішніх і внутрішніх метрик, складно, тому, як

правило, будується гіпотетична модель, взаємозв'язок метрик в якій моделюється статистично в ході використання метрик.

5.2 Проектування метрик якості

Для створення моделі якості ПЗ необхідно:

1. Визначити цілі вимірювання і виконувати всі наступні дії з урахуванням факторів, що впливають на якість.
2. Ідентифікувати компоненти ПЗ, якість яких повинна моделюватися окремо.
3. Виділити і класифікувати найбільш істотні компоненти, які несуть відповідальність за якість властивості (атрибути) для кожного компонента.
4. Сформулювати аксіоми для зв'язування властивостей ПЗ з характеристиками якості.
5. Визначити систему внутрішніх і зовнішніх метрик якості ПЗ, адекватну переліком характеристик якості і атрибутів ПЗ.
6. Перевірити модель якості.

Процес, що виконується при розробці власних або виборі існуючих метрик якості ПЗ, що відповідають цільовим вимогам до якості ПЗ (або її компоненту), містить наступні кроки:

Крок 1. Визначення понять. Визначення всіх використовуваних понять повинні відповідати зазначеним у застосовуваних стандартах або приводитися в документі, що описує прийняту модель якості ПЗ. Не можна допускати неоднозначності тлумачення термінів різними категоріями осіб, які використовують метрики (користувачами, замовниками, розробниками).

Крок 2. Визначення внутрішньої структури (моделі) кожної метрики. Значення базових метрик вимірюються безпосередньо і модель таких метрик – це найменування відповідної змінної. Значення складних метрик являють собою математичні моделі, що використовують базові або інші складні метрики. Переважний прагматичний підхід до моделювання метрик – метрики повинні відображати найбільш важливі аспекти вимірюваного атрибуту і не бути занадто складними. Моделі метрик можуть бути запозичені з стандартів, наукової літератури, з досвіду інших організацій і т.д. і

повинні накопичуватися і при необхідності адаптуватися до потреб конкретних вимірювань. Для розроблення нових (оригінальних) метрик корисно залучати експертів в проблемній області проекту та інженерії якості ПЗ.

Крок 3. Формулювання методу обчислення метрики (критерію оцінювання). Модель кожної метрики декомпонується до рівня метрик-примітивів (базових метрик) і далі для цих примітивів визначається механізм отримання значення (критерій оцінювання). Приклад метрики-примітиву зі складним механізмом отримання значення – SLOC.

Метрики-примітиви та критерії їх оцінювання утворюють перший рівень необхідних даних, що збираються.

Методи визначення значень метрик, рекомендовані стандартом ДСТУ 2850, такі:

- вимірювальний. Метод ґрунтується на отриманні інформації про властивості та характеристики ПЗ з використанням вимірювальних технічних і програмних засобів (розмір завантажувального модуля, час виконання гілки програми та ін.);

- реєстраційний. Метод ґрунтується на отриманні інформації під час випробувань або при безпосередньому використанні ПЗ за призначенням, коли реєструються і підраховуються певні події (моменти часу збоїв, кількість відмов і ін.);

- розрахунковий. Метод заснований на використанні теоретичних і емпіричних залежностей на ранніх стадіях розробки, а також статистичних даних, що накопичуються при випробуваннях, експлуатації та супроводі. Цим методом прогнозуються характеристики і підхарактеристики на ранніх стадіях розробки (точність, стійкість, надійність та ін.) Розрахунковий метод використовується і для визначення фактичних значень характеристик за результатами тестування;

- експертний. Метод полягає у визначенні значень характеристик групою фахівців-експертів. Застосовується в тому випадку, якщо завдання визначення значень не може бути вирішена іншими методами. Для проведення експертизи встановлюються контрольовані ознаки, що містяться у вигляді запитань в аналітичній опитувальній анкеті експертів.

Крок 4. Визначення критерію «хорошого» значення метрики. Коли відомо, що вимірювати і як вимірювати, потрібно визначити, як інтерпретувати результати вимірювань, і якими мають бути межі їх змін.

Критерієм істини у багатьох випадках служить думка замовника і користувача. Можуть також використовуватися опубліковані відомості про прийнятні значення тих чи інших метрик, отримані в галузевій виробничій практиці значення та ін. Так, наприклад, відомо, що Цикломатична складність модулів повинна бути ≤ 10 . Найбільш достовірне джерело інформації – власні історичні дані про вимір схожих проектів.

Крок 5. Документування метрик. На цьому кроці потрібно прийняти рішення про формат звіту, про цикл витягу та реєстрації даних (частоті вимірювань), про механізми обліку даних (тверда копія, електронна копія), про порядок їх розподілу (напрямки) різним учасникам процесу вимірювання, про обмеження (визначають готовність метрики до використання) та ін.

Крок 6. Визначення додаткових кваліфікаторів метрик. Метрика вважається добре спроектованою, якщо вона є узагальненням різних поглядів і рівнів вимірювання і якщо її можна застосовувати, наприклад, на рівні класу продуктів, одного продукту або компонентів цього продукту ПЗ. В якості додаткових кваліфікаторів може вказуватися контекст застосування й інші обмеження.

Існує безліч різних метрик, які можуть мати цінність для керування сучасним процесом.

Якісна метрика володіє наступними основними характеристиками:

1. Метрика має сенс для замовника, менеджера та виконавця. Якщо хоча б одна із зацікавлених сторін не розглядає метрику як осмислену, вона не буде використовуватися.

2. Метрика показує кореляцію між змінами в процесі розроблення та ходом бізнесу. Єдиними реальними цілями та завданнями організації є фінансові: зниження витрат, підвищення доходів і збільшення прибутку.

3. Метрика повинна бути об'єктивною та її визначення недвозначно. Об'єктивність слід трансформувати в певну форму кількісної уяви (наприклад, певні значення, відсотки, частки). Невизначеність мінімізується за допомогою використання зрозумілих

одиниць виміру (таких, як людино-місяці, SLOC, зміни, функціональні точки, класи, сценарії, вимоги), які не так просто визначити точно.

4. Метрика має показувати тенденції. Це дуже важлива характеристика. Розуміння змін значення метрики в залежності від часу, від проекту до проекту, від версії до версії і т.д. виявляється надзвичайно важливим, особливо для сучасних моделей ітераційного розроблення. Рідко трапляється, щоб деяка метрика безпосередньо приводила до виявлення яких-небудь конкретних дій. Швидше, метрика демонструє погляд з певної точки зору, та інтерпретувати цю метрику та визначати необхідні дії повинні менеджери проектів.

5. Метрика є природним побічним продуктом процесу. Це означає, що заради метрики не додаються нові робочі продукти або додаткові види діяльності; вона виводиться безпосередньо з самих робочих процесів.

6. Метрика може бути автоматизована. Досвід показує, що найбільш вдалим метриками є ті, інформація про які збирається та виявляється автоматично. Частково це так, тому що програмний інструментарій вимагає строгих визначень для даних, що оброблюються.

5.4 Практичні завдання

Завдання 1. Оберіть з переліку метрик базові: кількість рядків коду; кількість пустих рядків; кількість рядків, що містять; кількість рядків, що містять вихідний код і коментарі; кількість рядків, що містять декларативний вихідний код; кількість рядків, що містять імперативний вихідний код; відсоток коментарів; середня кількість рядків для функцій (методів); середня кількість рядків, що містять вихідний код для функцій (методів); середня кількість рядків для модулів; середня кількість рядків для класів, словник операторів (кількість унікальних операторів програми, також символи-роздільники, імена процедур і знаки операцій); словник операндів (кількість унікальних операндів програми); загальна кількість операторів в програмі; загальна кількість операндів в програмі;

довжина програми; обсяг програми; цикломатична складність; рівень вкладеності і довжини програми.

Завдання 2. Класифікуйте метрики перерахованні в завданні 1.

Завдання 3. Оберіть методи визначення значень для даних метрик (вимірювальний, реєстраційний, розрахунковий, експертний). Обґрунтуйте свій вибір

5.5 Контрольні запитання

1. Чим відрізняється метрика та міра якості?
2. Які шкали використовуються для вимірювання значень метрик?
3. Наведіть класифікацію мір якості.
4. Наведіть класифікацію метрик якості.
5. На чому ґрунтується розробка внутрішніх та зовнішніх метрик якості?
6. Наведіть кроки розробки власних метрик якості.
7. Якими характеристиками повинні володіти якісні метрики?

5.6 Література до розділу

Визначення метрик інформаційних систем: [Ошибка! Источник ссылки не найден.],[Ошибка! Источник ссылки не найден.-6],[21],[28-36],[41-43],[45],[50-51],[78-81],[91-100].

6 МЕТРИКИ КОДУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У загальному випадку застосування метрик дозволяє керівникам проєктів і підприємств вивчити складність розробленого або майбутнього проєкту, оцінити обсяг робіт, стилістику програми що розробляється і зусилля, витрачені кожним розробником для реалізації того чи іншого рішення. Однак метрики можуть служити лише рекомендаційними характеристиками, ними не можна повністю керуватися.

6.1 Кількісні метрики

Кількість рядків коду. Найпоширенішою метрикою вихідного коду, що відображає розмір програмного проєкту, є показник кількості рядків коду *SLOC* (*Source Lines Of Code*,). Не вдаючись в чергове перерахування недоліків даної метрики, розглянемо особливості її обчислення, а також існуючі різновиди та похідні.

Спочатку *SLOC* застосовувався в умовах, коли мови програмування були відносно прості за структурою і для них характерним було відповідність одного рядка коду одній команді мови. Згодом мови програмування еволюціонували, стали набагато гнучкіші, і ця відповідність для них більше не виконувалося – один фізичний рядок коду тепер міг містити кілька команд мови. З урахуванням цього виділяють і два основні різновиди показника *SLOC*:

– кількість фізичних *SLOC* (використовувані аббревіатури: *LOC*, *SLOC*, *KLOC*, *KSLOC*, *DSLOC*) – визначається як загальна кількість рядків коду, з урахуванням коментарів і порожніх рядків (при вимірюванні показника на кількість порожніх рядків, як правило, вводиться обмеження – враховується їх кількість, що не перевищує 25% загальної кількості рядків у вимірюваному блоці коду). Літера "D" (*Delivered*) означає "поставлений" код, тобто тільки той, що увійшов до закінченого програмного продукту. Дуже часто виникає

необхідність враховувати не тільки поставлений код, але і допоміжний або проміжний, який не входить до закінченого продукту, але що потрібен для реалізації проекту і вимагає певних витрат праці;

- кількість логічних *SLOC* (використовувані абревіатури: *LSI*, *DSI*, *KDSI*, де *SI* – *Source Instructions*) – визначається як кількість команд і залежить від використовуваної мови програмування. В тому випадку, якщо мова не допускає розміщення на одному рядку кількох команд, кількість логічних *SLOC* буде відповідати числу "фізичних за винятком порожніх рядків та рядків коментарів. Якщо ж мова програмування підтримує розміщення кількох команд на одному рядку, то один «фізичний» рядок повинний бути врахований як декілька «логічних», якщо він містить більше однієї команди мови, що використовується.

Нерідко показник *SLOC* використовується на ранніх стадіях роботи над проектом з метою отримання попередніх оцінок його загального обсягу. І хоча завдання це досить складне, а про отримання точних оцінок не може бути й мови, для його вирішення є спеціальні підходи, які забезпечують прийнятний результат:

- оцінка показника *SLOC* за аналогією. Дана методика дозволяє отримувати оцінку показника для всього проекту, що розробляється, або окремих його складових за допомогою порівняння функціональних властивостей з існуючими аналогами. При оцінці показника за даною методикою підбирається близький за функціональністю замітник з відомим значенням *SLOC*, далі на його підставі визначається прогнозна оцінка з урахуванням відмінностей у функціональності, у мовах програмування, можливості оптимізації на основі коду аналога й інш.;

- оцінка показника *SLOC* «знизу-вгору» експертним методом. Цією методикою передбачено розподіл проекту на ієрархічну структуру завдань (*Work Breakdown Structure, WBS*), на підставі яких виконується експертне оцінювання показника, та підсумовується в результаті для всього проекту. Як правило, експертами надаються три оцінки (найбільш вірогідна, максимальна і мінімальна), а далі вони усереднюються за допомогою бета-розподілу (розраховується сума мінімальної, максимальної і чотириразової найбільш імовірної оцінки, яка потім ділиться на шість).

Для метрики *SLOC* є багато похідних метрик, що дозволяють оцінити окремі показники проекту. Основними серед яких є:

- кількість пустих рядків (*Blank Lines Of Code, BLOC*);
- кількість рядків, що містять коментарі (*Comment Lines Of Code, CLOC*);
- кількість рядків, що містять вихідний код і коментарі (*Lines with Both Code and Comments, C&SLOC*);
- кількість рядків, що містять декларативний вихідний код;
- кількість рядків, що містять імперативний вихідний код;
- відсоток коментарів (кількість рядків коментарів, помножене на 100 і поділена на кількість рядків коду);
- середня кількість рядків для функцій (методів);
- середня кількість рядків, що містять вихідний код для функцій (методів);
- середня кількість рядків для модулів;
- середня кількість рядків для класів.

Не зважаючи на всі недоліки цієї метрики, вона досі є ефективним інструментом вимірювання якості та є складовою багатьох інших метрик.

Стилістика програми. Іноді важливо не просто підрахувати кількість рядків коментарів, а визначити їх щільність, тобто визначити якість документованості коду. Для оцінки **стилістики програми** (F), використовують вираз

$$F = \sum_{i=1}^n F_i, \quad (6.1)$$

де $F_i = \text{SIGN}(N_{\text{комм.и}}/N_i - 0,1)$ – стилістика i -го фрагмента програми, де $N_{\text{комм.и}}$ – кількість коментарів у i -м фрагменті, N_i – загальна кількість рядків коду в i -му фрагменті.

Метрики Холстеда. Моріс Холстед (Maurice H. Halstead) у 1977 запропонував групу метрик, побудованих на аналізі кількості рядків і синтаксичних елементів вихідного коду програми.

Оцінка складності програми (Halstead Difficulty, HDiff) оцінюється як:

$$Hdiff = (NUOprtr / 2) \times (NOprnd / NUOprnd), \quad (6.2)$$

де $NUOprtr$ – словник операторів (кількість унікальних операторів програми, також символи-роздільники, імена процедур і знаки операцій);

$NUOprnd$ – словник операндів (кількість унікальних операндів програми);

$Noprtr$ – загальна кількість операторів в програмі;

$Noprnd$ – загальна кількість операндів в програмі.

На основі (6.2) пропонується оцінювати зусилля програміста при розробці за допомогою показника $HEff$ (Halstead Effort):

$$HEff = HDiff \times HPVol, \quad (6.3)$$

де $HPVoc = NUOprtr + NUOprnd$ – словник програми (Halstead Program Vocabulary, $HPVoc$).

Крім того, для оцінювання програми можливо використовувати:
– довжина програми (Halstead Program Length, $HPLen$):

$$HPLen = Noprtr + Noprnd;$$

– обсяг програми (Halstead Program Volume, $HPVol$):

$$HPVol = HPLen \times \log_2 HPVoc.$$

Незважаючи на теоретичну можливість отримання попередніх оцінок трудомісткості проекту з використанням показника $HEff$, метрики Холстеда, як правило, обчислюються вже на основі готового вихідного коду і застосовуються для отримання апостеріорних оцінок. Вони можуть бути використані для того, щоб запобігти надмірному ускладненню окремих структурних елементів програмного проекту і зростання пов'язаних із цим ризиків.

Метрики Джилба. Ці метрики показують складність програмного забезпечення на основі насиченості програми умовними операторами або операторами циклу. Дана метрика, не дивлячись на

свою простоту, досить добре відображає складність написання і розуміння програми, а при додаванні такого показника, як максимальний рівень вкладеності умовних і циклічних операторів, ефективність цієї метрики значно зростає.

Абсолютна складність за Джилбом:

$$Cl = \sum N_{ifoprtr} , \quad (6.4)$$

де $N_{ifoprtr}$ – кількість умовних операторів.

Відносна складність за Джилбом:

$$Cl^* = \frac{CL}{N_{oprtr}} , \quad (6.5)$$

де N_{oprtr} – загальна кількість операторів в програмі.

6.3 Метрики складності потоку керування програми

Наступний великий клас метрик, заснований вже не на кількісних показниках, а на аналізі керуючого графа програми, називається метрики складності потоку керування програм.

Перед тим як безпосередньо описувати метрики, для кращого розуміння буде наведемо визначення керуючого графу програми і спосіб його побудови.

Нехай представлена певна програма. Для даної програми будується орієнтований граф, що містить лише один вхід і один вихід, при цьому вершини графа співвідносять з тими ділянками коду програми, в яких є лише послідовні обчислення і відсутні оператори розгалуження і циклу, а дуги співвідносять з переходами від блоку до блоку і гілками виконання програми. Умова при побудові даного графа: кожна вершина досяжна з початкової, і кінцева вершина досяжна з будь-якої іншої вершини.

Цикломатична складність програми. Найпоширенішою оцінкою, заснованою на аналізі отриманого графа, є цикломатичне число Мак-Кейба, що визначається як:

$$V(G) = e - n + 2p, \quad (6.6)$$

де e – кількість дуг, n – кількість вершин, p – кількість компонентів зв'язності. Кількість компонентів зв'язності графа можна розглядати як кількість дуг, що необхідно додати для перетворення графа в сильно зв'язний. Сильно зв'язним називається граф, будь-які дві вершини якого взаємно досяжні. Для графів коректних програм, тобто графів, що не мають недосяжних від точки входу ділянок і "вісячих" точок входу і виходу, сильно зв'язний граф, як правило, виходить шляхом замикання дугою вершини, що означає кінець програми, на вершину, що позначає точку входу в цю програму. По суті $V(G)$ визначає кількість лінійно незалежних контурів в сильно доладному графі. Так що в коректно написаних програмах $p=1$, і тому формула (6.6) для розрахунку цикломатичної складності набуває вигляду:

$$V(G) = e - n + 2. \quad (6.7)$$

Як правило, при обчисленні цикломатичної складності логічні оператори не приймаються до уваги, допускається також спрощений підхід, згідно з яким власне побудова графа не проводиться, а показник визначається на підставі підрахунку кількості операторів керуючої логіки (*if*, *switch* і т. д.) і можливої кількості шляхів виконання програми. Метрика цикломатичної складності може бути розрахована для модуля, методу та інших структурних одиниць програми.

В процесі аналізу значень показника для окремих структурних елементів можна виявити елементи з високим значенням показника (наприклад, нормальне значення показника для методу – не вище 5-7), що свідчить про складність їх керуючої логіки і, відповідно, високих трудовитрат на розробку, тестування і супровід.

Обчислення метрики в ході реалізації проекту (а при детальному проектуванні воно можливо ще на цьому етапі, не чекаючи стадії

кодування) дозволяє вчасно визначити найбільш складні, що супроводжуються високими ризиками, структурні одиниці та вжити заходів щодо усунення ризиків за рахунок внесення корективів.

Для метрики цикломатичної складності існує безліч варіацій, зокрема:

- "модифікована" цикломатична складність – розглядає не кожне розгалуження оператора множинного вибору (switch, case), а весь оператор як єдине ціле;

- "сувора" цикломатична складність – містить логічні оператори;

- "спрощена" цикломатична складність – передбачає обчислення на основі не графа, а підрахунку керуючих операторів;

- "актуальна" складність – визначається як кількість незалежних шляхів, що проходить програма при тестуванні;

- метрика складності глобальних даних – обчислюється як цикломатична складність модуля та збільшується на кількість взаємозв'язків з глобальними даними.

В цілому, метрики цикломатичної складності є досить гарним показником, що дозволяє своєчасно припинити подальше ускладнення окремих складових проекту і спростити їх, попередивши ймовірні проблеми з заплутаним і нестабільним кодом в майбутньому.

Метрики похідні від цикломатичної складності. Метрика цикломатичної складності стала основою для створення похідних і якісно нових метрик, таких як:

- інтервальна метрика Дж. Майерса (G. Mayers) – розраховується як різниця значень цикломатичної складності й кількості окремих умов плюс одиниця;

- метрика У. Хансена (W. Hansen) – визначається парою "циклوماتична складність" і "кількість операторів";

- метрика Пивоварського – обчислюється на основі модифікованої цикломатичної складності і дозволяє враховувати структурованість програми, виявляючи код з поганою структурою, і багатьох інших, нерідко мають швидше за наукову, ніж практичну цінність:

$$N(G_i) = V^*(G_i) + \sum P_i, \quad (6.8)$$

де $V^*(G)$ – модифікована цикломатична складність, обчислена так як і $V(G)$, але з однією відмінністю: оператор CASE з n виходами розглядається як один логічний оператор, а не як $n - 1$ операторів;

P_i – глибина вкладеності i -ї предикатної вершини. Для підрахунку глибини вкладеності предикатних вершин використовується число "сфер впливу". Під глибиною вкладеності розуміється кількість всіх "сфер впливу" предикатів, які повністю утримуються у сфері даної вершини, або перетинаються з нею. Глибина вкладеності збільшується за рахунок вкладеності не самих предикатів, а "сфер впливу". Міра Пивоварського зростає при переході від послідовних програм до вкладених і далі до неструктурованих, що є її величезною перевагою перед багатьма іншими заходами даної групи.

Топологічна міра Чена. Ця метрика висловлює складність програми через кількість перетинів кордонів між областями, утвореними графом програми:

$$M(G_i) = (n(G_i), N, Q_0), \quad (6.9)$$

Цей підхід застосовується лише до структурованих програм, що допускає лише послідовне з'єднання керуючих конструкцій. Для неструктурованих програм міра Чена істотно залежить від умовних і безумовних переходів. У цьому випадку можна вказати верхню і нижню границі. Верхня – є $m+1$, де m – кількість логічних операторів при їх взаємній вкладеності. Нижня – дорівнює 2. Коли керуючий граф програми має тільки одну компоненту зв'язності, захід Чена збігається з цикломатичною мірою Мак-Кейба.

Метрики Харрісона і Мейджела. Ці метрики враховують рівень вкладеності і довжини програми.

Кожній вершині присвоюється своя складність у відповідності з оператором, що вона зображує. Ця початкова складність вершини може обчислюватися будь-яким способом, включаючи використання заходів Холстеда. Виділимо для кожної предикатної вершини підграф, породжений вершинами, які є кінцями вихідних з неї дуг, а також вершинами, досяжними з кожної такої вершини (нижня межа підграфа), і вершинами, що лежать на шляхах з предикатної вершини

в яку-небудь нижню межу. Цей підграф називається сферою впливу предикатної вершини.

Приведеною складністю предикатної вершини називається сума початкових або наведених складнощів вершин, що входять в її сферу впливу, плюс первинна складність самої предикатної вершини.

Функціональна міра (*SCOPE*) програми:

$$f_i = \sum \chi_i, \quad (6.10)$$

де χ_i – це складність вершини керуючого графа.

Функціональним ставленням (*SCORT*) називається відношення числа вершин в керуючому графі до його функціональної складності, причому з числа вершин виключаються термінальні:

$$f_i^* = \frac{N\chi_i}{f_i}, \quad (6.11)$$

де N – це кількість вершин керуючого графа.

SCORT може приймати різні значення для графів з однаковим цикломатичним числом.

Метод граничних значень так само заснований на аналізі керуючого графа програми. Для визначення даного методу необхідно ввести кілька додаткових понять.

Нехай $G = (V, E)$ – керуючий граф програми з єдиною початковою і єдиною кінцевою вершинами.

Кількість вершин, що входять до дуг, називається негативним ступенем вершини, а кількість вихідних з вершини дуг – позитивним ступенем вершини. Тоді набір вершин графа можна розбити на дві групи: вершини, у яких позитивна ступінь ≤ 1 ; вершини, у яких позитивна ступінь ≥ 2 .

Вершини першої групи назвемо приймаючими вершинами, а вершини другої групи – вершинами відбору.

Кожна приймаюча вершина має наведену складність, рівну 1, крім кінцевої вершини, наведена складність якої дорівнює 0. Наведені складності всіх вершин графа G підсумовуються, утворюючи

абсолютну граничну складність програми. Після цього визначається відносна гранична складність програми:

$$S_g = 1 - (v - 1) / S_a, \quad (6.12)$$

де S_g – відносна гранична складність програми, S_a – абсолютна гранична складність програми, v – загальна кількість вершин графа програми.

Існує *метрика Шнейдевінда*, що виражається через кількість можливих шляхів в керуючому графі.

Міра Вудворда. В основі цієї метрики кількість перетинів дуг керуючого графа. Так як в добре структурованій програмі таких ситуацій виникати не повинно, то дана метрика застосовується в основному в слабо структурованих мовах (Асемблер, Фортран). Точка перетину виникає при виході керування за межі двох вершин, які є послідовними операторами.

6.4 Метрики складності потоку керування даними

Наступний клас метрик – метрики складності потоку керування даних.

Метрика Чепина. Суть методу полягає в оцінці інформаційної міцності окремо взятого програмного модуля з допомогою аналізу характеру використання змінних зі списку введення-виведення.

Вся безліч змінних, що складають перелік введення-виведення, розбивається на 4 функціональні групи :

- змінні, що використовуються для розрахунків і для забезпечення виведення (P);
- модифіковані, або створювані всередині програми змінні (M);
- «керуючі» змінні – змінні, які беруть участь в керуванні роботою програмного модуля (C),
- "паразитні" змінні – змінні не використовуються в програмі, (T).

Оскільки кожна змінна може виконувати одночасно кілька функцій, необхідно враховувати її в кожній відповідній функціональній групі.

Отже метрика Чепина має вигляд:

$$Q = a_1 * P + a_2 * M + a_3 * C + a_4 * T, \quad (6.13)$$

де a_1, a_2, a_3, a_4 – вагові коефіцієнти, що відображають різний вплив на складність програми кожної функціональної групи. На думку автора метрики, найбільшу вагу, рівну 3, має функціональна група C , так як вона впливає на потік керування програми. Вагові коефіцієнти інших груп розподіляються наступним чином: $a_1=1, a_2=2, a_4=0.5$. Ваговий коефіцієнт групи T не дорівнює 0, оскільки "паразитні" змінні не збільшують складність потоку даних програми, але іноді ускладнюють її розуміння. З урахуванням вагових коефіцієнтів:

$$Q = P + 2M + 3C + 0.5T. \quad (6.14)$$

Метрика Спена. Ця метрика ґрунтується на локалізації звернень до даних усередині кожної програмної секції. **Спен** – це кількість тверджень, які містять цей ідентифікатор, між першою і останньою появою в тексті програми. Отже, ідентифікатор, що з'явився n раз, має спен, рівний $n-1$. При великому спені ускладнюється тестування та налагодження.

Метрика "модуль – глобальна змінна". Ця метрика, що зв'язує складність програм із зверненнями до глобальних змінних. Пара "модуль-глобальна змінна" позначається як (p,r) , де p – модуль, що має доступ до глобальної змінної r . У залежності від наявності у програмі реального звернення до змінної r формуються два типи пар "модуль – глобальна змінна": фактичні та можливі. Можливе звернення до r за допомогою p показує, що область існування r містить у собі p .

Дана характеристика позначається A_{up} і говорить про те, скільки разів модулі up дійсно отримували доступ до глобальних змінних, а число P_{up} – скільки разів вони могли б отримати доступ.

Відношення кількості фактичних звернень до можливих визначається як:

$$R_{up} = A_{up}/P_{up}. \quad (6.15)$$

Ця формула показує наближену ймовірність посилення довільного модуля на довільну глобальну змінну. Очевидно, що чим вище ця ймовірність, тим вище ймовірність "несанкціонованої" зміни будь-якої змінної, що може істотно ускладнити роботу, пов'язану з модифікацією програми.

Міра Кафура. Вона створена на основі концепції інформаційних потоків. Для використання цієї міри вводяться поняття локального та глобального потоку.

Локальний потік інформації з модуля A до модуля B існує, якщо:

- модуль A викликає модуль B (прямий локальний потік);
- модуль B викликає модуль A і модуль A повертає значення, що використовується у модулі B (непрямий локальний потік)
- модуль C викликає модулі A і B та передає результат виконання модуля A до модуля B .

Глобальний потік інформації з модуля A через глобальну структуру даних D існує, якщо модуль A розміщує інформацію в D , а модуль B використовує інформацію з D .

На основі цих понять вводиться величина I – **інформаційна складність процедури**:

$$I = length * (fan_in * fan_out)^2, \quad (6.16)$$

де $length$ – складність тексту процедури (вимірюються через метрику обсягу, типу метрик Холстеда (6.2-6.3), Маккейба (6.6), LOC і т.п.); fan_in – кількість локальних потоків входять всередину процедури плюс число структур даних, з яких процедура бере інформацію fan_out – кількість локальних потоків вихідних з процедури плюс число структур даних, які оновлюються процедурою.

Можна визначити інформаційну складність модуля як суму інформаційних складнощів процедур, що входять до нього.

Далі розглянемо інформаційну складність модуля щодо певної структури даних. Інформаційна міра складності модуля щодо структури даних:

$$J = N_w * N_R + N_w * N_{RW} + N_{RW} * N_R + N_{RW} * (N_{RW} - 1), \quad (6.17)$$

де N_W – кількість процедур, що тільки оновлюють структуру даних; N_R – кількість процедур, що тільки читають інформацію з структури даних; N_{RW} – кількість процедур, що і читають, і оновлюють інформацію в структурі даних.

Міра Овієдо. Суть її полягає в тому, що програма розбивається на лінійні непересічні ділянки – промені операторів, які утворюють керуючий граф програми.

Автор метрики виходить з таких припущень: програміст може знайти відношення між визначеними та використаними входженнями змінної всередині променя більш легко, ніж між променями; число різних визначальних записів у кожному промені більш важливо, ніж загальне число використовуваних входжень змінних в кожному промені.

Позначимо через $R(i)$ безліч визначальних входжень змінних, які розташовані в радіусі дії променя i (визначальне входження змінної знаходиться в радіусі дії променя, якщо змінна або локальна в ньому і має визначальне входження, або для неї є визначальне входження в деякому попередньому промені, і немає локального визначення по шляху). Позначимо через $V(i)$ множину змінних, які використовують входження і вже є в промені i . Тоді міра складності i -го променя визначається як:

$$DF(i) = \sum (DEF(v_j)), \quad (6.18)$$

де $DEF(v_j)$ – визначає кількість входжень змінної v_j з безлічі $R(i)$, $j = 1 \dots //V(i)//$, а $//V(i)//$ – потужність безлічі $V(i)$.

6.5 Метрики складності потоку керування і даних програми

Четвертим класом метрик є метрики, близькі як до класу кількісних метрик, класу метрик складності потоку керування програми, так і до класу метрик складності потоку керування даними (строго кажучи, цей клас метрик і клас метрик складності потоку керування програми є одним і тим же класом – топологічними

метриками, але має сенс розділити їх у даному контексті для більшої ясності). Даний клас метрик встановлює складність структури програми як на основі кількісних підрахунків, так і на основі аналізу структур керування.

Тестуюча M-Mіра. Тестуючою мірою M називається міра складності, що задовольняє наступним умовам:

1. Міра складності простого оператора дорівнює 1.
2. $M(\{F_1; F_2; \dots; F_N\}) = e_i^n M(F_p)$
3. $M(\text{if } p \text{ then } f1 \text{ else } f2) = 2 \max(M(F1), M(F2))$
4. $M(\text{while } p \text{ do } F) = 2M(F)$.

Міра зростає з глибиною вкладеності й враховує протяжність програми. До тестуючої міри близько примикає міра на основі регулярних вкладень. Ідея цієї міри складності програм полягає в підрахунку сумарної кількості символів (операндів, операторів, дужок) у регулярному виразі з мінімально необхідною кількістю дужок, що описує керуючий граф програми. Всі міри цієї групи чутливі до вкладеності керуючих конструкцій і до протяжності програми. Проте зростає рівень трудомісткості обчислень.

Зв'язаність модулів програми. Якщо модулі сильно пов'язані, то програма стає важкомодифікуємою і важкою в розумінні. Дана міра не виражається кількісно.

Види зв'язаності модулів:

- зв'язаність за даними – якщо модулі взаємодіють через передавання параметрів і при цьому кожен параметр є елементарним інформаційним об'єктом. Це найбільш бажаний тип зв'язаності (зчеплення);
- зв'язаність за структурою даних – якщо один модуль посилає іншому складовою інформаційний об'єкт (структуру) для обміну даними;
- зв'язаність за керуванням – якщо один модуль посилає іншому інформаційний об'єкт – флаг, призначений для керування його внутрішньою логікою;

- зв'язаність за загальною областю, якщо модулі посилаються на одну і ту ж область глобальних даних. Зв'язаність (зчеплення) за загальною областю є небажаним, оскільки, по-перше, помилка в модулі, що використовує глобальну область, може несподівано проявитися в будь-якому іншому модулі; по-друге, такі програми важкі для розуміння, так як програмісту важко визначити, які саме дані використовуються конкретним модулем;

- зв'язаність за вмістом – якщо один з модулів посилається всередину іншого. Це неправильний тип зчеплення, оскільки повністю суперечить принципу модульності, тобто подання модуля у вигляді чорного ящика. Зовнішня зв'язаність – два модуля використовують зовнішні дані, наприклад комунікаційний протокол.

- зв'язаність за допомогою повідомлень – найбільш вільний вигляд зв'язаності, модулі безпосередньо не пов'язані один з одним, і взаємодіють через повідомлення, що не мають параметрів.

- підкласова зв'язаність – відношення між класом-батьком і класом-нащадком, причому нащадок пов'язаний з батьком, а батько з нащадком – ні.

- зв'язаність за часом – два дії згруповані в одному модулі лише тому, що через обставини вони відбуваються в один час.

Відсутність зв'язаності – модулі не взаємодіють між собою.

Міра Колофелло. Ще одна міра, що стосується стабільності модуля $h(G)$. вона може бути визначена як кількість змін, які потрібно зробити в модулях, відмінних від модуля, стабільність якого перевіряється, при цьому ці зміни повинні стосуватися модуля, що перевіряється.

Метрика Мак-Клура. Виділяються три етапи обчислення цієї метрики:

1. Для кожної змінної обчислюється значення її функції складності $C(i)$ за формулою:

$$C(i) = (D(i) * J(i)) / n, \quad (6.19)$$

де $D(i)$ – величина, що стосується сфери дії змінної i . $J(i)$ – міра складності взаємодії модулів через змінну i , n – кількість окремих модулів у схемі розподілу.

2. Для всіх модулів, що входять у сферу розбиття, визначається значення їх функцій складності $M(P)$ за формулою:

$$M(P) = fp * X(P) + gp * Y(P), \quad (6.20)$$

де fp і gp – відповідно, кількість модулів, безпосередньо передують і безпосередньо наступних за модулем P , $X(P)$ – складність звернення до модулю P , $Y(P)$ – складність керування викликом з модулю P інших модулів.

3. Загальна складність MP ієрархічної схеми розподілу програми на модулі задається формулою:

$$MP = \sum M(P) \quad (6.21)$$

Для всіх можливих значень P – модулів програми. Дана метрика орієнтована на добре структуровані програми, складені з ієрархічних модулів, які задають функціональну специфікацію і структуру керування. Також мається на увазі, що в кожному модулі одна точка входу і одна точка виходу, модуль виконує рівно одну функцію, а виклик модулів здійснюється у відповідності з ієрархічною системою керування, яка задає ставлення виклику на безлічі модулів програми.

Міра Берлінгера. Метрика, заснована на інформаційній концепції. Міра складності обчислюється як:

$$M = \sum f_i * \log_2 p_i, \quad (6.22)$$

де f_i – частота появи i -го символу, p_i – імовірність його появи.

Недоліком цієї метрики є те, що програма, що містить багато унікальних символів, але в малій кількості, буде мати таку ж складність як програма, що містить невелику кількість унікальних символів, але у великій кількості.

6.6 Об'єктно-орієнтовані метрики

У зв'язку з розвитком об'єктно-орієнтованих мов програмування з'явився новий клас метрик, також названий об'єктно-орієнтованими (ОО) метриками. У даній групі найбільш часто використовуваними є набори метрик Мартіна і набір метрик Чидамбера і Кемерера.

Метрики Мартіна. Для використання цих метрик необхідно ввести поняття категорії класів. У реальності клас може досить рідко бути використаний повторно ізольовано від інших класів. Практично кожен клас має групу класів, з якими він працює в кооперації, і від яких він не може бути легко відокремлений. Для повторного використання таких класів необхідно повторно використовувати всю групу класів. Така група класів сильно пов'язана і називається **категорією класів**. Для існування категорії класів існують наступні умови:

- класи в межах категорії класу закриті від будь-яких спроб змін всі разом. Це означає, що, якщо один клас повинен змінитися, всі класи в цій категорії з великою ймовірністю зміняться. Якщо будь-який з класів відкритий для деякого різновиду змін, вони всі відкриті для такого різновиду змін;

- класи у категорії повторно використовуються тільки разом. Вони настільки взаємозалежні і не можуть бути відокремлені один від одного. Таким чином, якщо робиться будь-яка спроба повторного використання одного класу в категорії, всі інші класи повинні повторно використовуватися з ним;

- класи у категорії поділяють деяку загальну функцію або досягають деякої спільної мети. Відповідальність, незалежність і стабільність категорії можуть бути виміряні шляхом підрахунку залежностей, які взаємодіють з цією категорією.

Для категорій класів можуть бути визначені три метрики :

1. Доцентрове зчеплення (C_a) – кількість класів поза певної категорії, що залежать від класів всередині цієї категорії.
2. Відцентрове зчеплення (C_e) – кількість класів всередині певної категорії, що залежать від класів поза цієї категорії.
3. Нестабільність:

$$St = Ce / (Ca + Ce). \quad (6.23)$$

Ця метрика має діапазон значень $[0,1]$. Якщо $St = 0$, це вказує максимально стабільну категорію, а якщо $St = 1$, це вказує максимально нестабільну категорію.

Абстрактність категорії визначається наступним чином:

$$A = N_A / N_{All},$$

де N_A – кількість абстрактних класів в категорії, N_{All} – спільна кількість класів в категорії.

Значення цієї метрики змінюються в діапазоні $[0,1]$, якщо $A=0$, то категорія цілком конкретна, а якщо $A=1$ – категорія повністю абстрактна.

Тепер на основі наведених метрик Мартіна можна побудувати графік, на якому відображено залежність між абстрактністю і нестабільністю. Якщо на ньому побудувати пряму, що задається формулою $1+A=1$, то на цій прямій будуть лежати категорії, що мають кращу збалансованість між абстрактністю і нестабільністю. Ця пряма називається головною послідовністю.

Далі можна ввести ще 2 метрики:

Відстань до головної послідовності:

$$D = (A + St - 1) / \sqrt{2}$$

Нормалізованої відстань до головної послідовності:

$$Dn = A + St - 2$$

Практично для будь-яких категорій вірно те, що чим ближче вони знаходяться до головної послідовності, тим краще.

Загальні об'єктно-орієнтовані метрики. До найбільш поширених *ОО* метрик відносяться:

– кількість виклику віддалених методів (*Number Of Remote Methods, NORM*). При її обчисленні проглядаються всі конструктори і методи класу і підраховується кількість викликів віддалених методів (не визначених у класі та його батьків);

– відгук на клас (*Response For Class, RFC*) – кількість методів, які можуть викликатися примірниками класу, обчислюється як сума локальних і віддалених методів;

– зважена насиченість класу 1 (*Weighted Methods Per Class 1, WMPC1*) – відображає відносну міру складності класу на основі цикломатичної складності кожного його методу. Клас з більш складними методами і великою їх кількістю вважається більш складним (при обчисленні метрики батьківські класи не враховуються);

– зважена насиченість класу 2 (*WMPC2*) – міра складності класу, заснована на тому, що клас з більшою кількістю методів і метод з більшою кількістю параметрів є більш складними (при обчисленні метрики батьківські класи не враховуються);

– недолік зв'язності методів 1 (*Lack Of Зчеплення Methods Of 1, LOCOM1*) – відображає міру взаємозалежності методів (зв'язність характеризує ступінь взаємодії між елементами окремого модуля і його насиченість). Чим вище значення даної метрики, тим більше взаємозалежними і, відповідно, складними вважаються методи;

– недолік зв'язності методів 2 (*LOCOM2*) – відображає міру взаємозалежності класів. Обчислюється як процентне відношення методів, що не мають доступу до специфічних атрибутів класу, до всіх атрибутів даного класу. Клас, що може викликати значно більше методів, ніж рівні йому за рівнем, є більш складним;

– недолік зв'язності методів 3 (*LOCOM3*) – вимірює ступінь розходження методів у класі за атрибутами. Низьке значення свідчить про хорошу декомпозицію в класі, що виражається в його простоті, зрозумілості та готовності до повторного використання. Високе значення – нестачу зв'язності, що збільшує складність і підвищує ймовірність помилок в процесі розроблення.

Метрики Чидамбера і Кемерера. Ці метрики засновані на аналізі методів класу, дерева успадкування і т.д.

Метрика *WMC* (*Weighted methods per class*), сумарна складність всіх методів класу:

$$WMC = \sum_{i=1}^n c_i,$$

де c_i – складність i -го методу, обчислена за будь-якою з метрик (Холстеда і т.д. в залежності від потрібного критерію), якщо у всіх методів складність однакова, то WMC дорівнює n .

Метрика *DIT* (*Depth of Inheritance tree*) – глибина дерева спадкування (найбільший шлях по ієрархії класів до цього класу від класу-предка), чим більше, тим краще, так як при більшій глибині збільшується абстракція даних, зменшується насиченість класу методами, однак при досить великій глибині сильно зростає складність розуміння і написання програми.

Метрика *NOC* (*Number of children*) – кількість нащадків (безпосередніх), чим більше, тим вище абстракція даних.

Метрика *CBO* (*Coupling between object classes*) – зчеплення між класами, показує кількість класів, з якими пов'язаний вихідний клас. Для даної метрики справедливі всі твердження, введені раніше для зв'язаності модулів, тобто при високому *CBO* зменшується абстракція даних і погіршується повторне використання класу.

Метрика *RFC* (*Response for a class*) – $RFC = RS$, де RS – відповідна множина класу, тобто безліч методів, які можуть бути потенційно викликані методом класу у відповідь на дані, отримані об'єктом класу. Тобто $RS = (\{M\}(\{R_i\}))$, $i = 1 \dots n$, де M – всі можливі методи класу, R_i – всі можливі методи, які можуть бути викликані i -м класом. Тоді *RFC* буде потужністю цієї множини. Чим більше *RFC*, тим складніше тестування та налагодження.

Метрика *LCOM* (*Lack of cohesion in Methods*) – недолік зчеплення методів. Для визначення цього параметра розглянемо клас C з n методами $M_1, M_2, \dots, M_n$, тоді $\{I_1\}, \{I_2\}, \dots, \{I_n\}$ – безлічі змінних, що використовуються в цих методах. Тепер визначимо P – безліч пар методів, що не мають спільних змінних; Q – безліч пар методів, що мають загальні змінні.

Тоді

$$LCOM = P / - / Q / .$$

Кількість пар методів розраховується як

$$C_m^2 = \frac{m!}{2(m-2)!},$$

де m – кількість методів класу.

Недолік зчеплення може бути сигналом того, що клас можна розбити на кілька інших класів або підкласів, так що для підвищення інкапсуляції даних і зменшення складності класів і методів краще підвищувати зчеплення.

6.7 Гібридні метрики

У завершенні необхідно згадати ще один клас метрик, званих гібридними. Метрики даного класу ґрунтуються на більш простих метриках і являють собою їх зважену суму.

Метрика Коккола. Вона визначається наступним чином:

$$H_M = (M + R1 * M(M1) + \dots + Rn * M(Mn)) / (1 + R1 + \dots + Rn)$$

де M – базова метрика, Mi – інші цікаві підходи, Ri – коректно підібрані коефіцієнти, $M(Mi)$ функції.

Функції $M(Mi)$ і коефіцієнти Ri обчислюються за допомогою регресійного аналізу або аналізу завдання для конкретної програми.

В результаті досліджень, автор метрики виділив три моделі для підходів: Маккейба, Холстеда і *SLOC*, де в якості базової використовується міра Холстеда. Ці моделі отримали назву "краща", "випадкова" і "лінійна".

Метрика Зольновського, Сіммонса, Тейера. Вона також являє собою зважену суму різних індикаторів. Існують два варіанти даної метрики:

- (структура, взаємодія, обсяг, дані) СУМА(a, b, c, d).
- (складність інтерфейсу, обчислювальна складність, складність введення/виводу, читабельність) СУМА(x, y, z, p).

Використовувані метрики в кожному варіанті вибираються залежно від конкретної задачі, коефіцієнти – в залежності від значення метрики для прийняття рішення в даному випадку.

Висновок:

Підводячи підсумок, хотілося б відзначити, що жодної універсальної метрики не існує. Будь-які контрольовані метричні характеристики програми повинні контролюватися або залежно один від одного, або залежно від конкретної задачі, крім того, можна застосовувати гібридні заходи, однак вони так само залежать від більш простих метрик і також не можуть бути універсальними. Строго кажучи, будь-яка метрика – це лише показник, який сильно залежить від мови і стилю програмування, тому ні одну міру не можна зводити в абсолют і приймати будь-які рішення, ґрунтуючись тільки на ній.

6.8 Практичні завдання

Завдання 1. Оберіть проект з архіву Microsoft – адреса сторінки <http://archive.msdn.microsoft.com/vcsamples>. Оберіть метрики та виконайте тестування програмних систем в IBM Rational ClearCase за їх допомогою:

1.1 Створіть скрипт обчислення метрик та вкажіть шлях до нього.

1.2 Додайте до контекстного меню елементів версійного сховища ClearCase елемент меню «Розрахунок метрик», з якого запускається скрипт обчислення метрик.

1.3 Виберіть проект у сховищі та запустіть скрипт на виконання.

Завдання 2. Виконати аналіз отриманих результатів з тестування.

6.9 Контрольні запитання

1. Які метрики відносяться до кількісних метрик?
2. Для розрахунку яких метрик необхідно зконструювати керуючий граф програми?
3. Які метрики є похідними від цикломатичної складності?
4. Порівняйте метрики складності потоку керування даними.
5. Для яких метрик використовується поняття «категорії класів»?

6. Що полягає в основі метрик Чидамбера та Кемерера?
7. Що аналізують об'єктно-орієнтовані метрики?

6.10 Література до розділу

Метрики коду: [6],[20],[42],[45-46],[47],[50],[51],[53],[95],[100].

7 МЕТРИКИ ЯКОСТІ ПРОГРАМНИХ ПРОЕКТІВ

Розглянемо метрики, які слід використовувати у всіх проектах зі створення ПЗ. Їх можна розділити на метрики керування та якості.

Показники якості керування:

- робота і прогрес (робота, що виконана на конкретний момент);
- фінансовий стан (витрати, понесені на конкретний момент);
- динаміка найму персоналу (зміни у складі персоналу, що відбулися на конкретний момент).
- середовище розроблення (*Software Engineering Environment*);
- використання системних ресурсів (*System Resource Utilization*).

Зазначені метрики можуть використовуватися на етапах планування та контролю проектів та інших завдань керування або використовуватися в якості параметрів керування штатної *ERP* системи.

Метрика «використання системних ресурсів» – визначає відсоток цільових комп'ютерних ресурсів, що використовуються системою.

«Середовище розроблення» – міра здатності виробника розробляти програмне забезпечення високої якості. Дана метрика може бути виражена в термінах моделі «*Software Acquisition Capability Maturity Model*» (*SA – CMM*).

Показники якості розробки:

- інтенсивність змін і стабільність (інтенсивність змін на визначений момент);
- дефекти та коефіцієнт дефективності (середня кількість дефектів на одну зміну на цей момент);
- доробки та адаптованість (середній обсяг доробок на одну зміну на цей момент);
- середній час напрацювання на відмову (*Mean time between failures, MTBF*) і завершеність (рівень дефектів на цей момент).
- складність інтерфейсів і інтеграції (*complexity of interfaces and integration*) – метрика, яка вимірює ступінь складності інтерфейсу або додаткового програмування необхідного для інтеграції компоненти в

систему, які потрібні для тестування, налагодження та супроводження, яке компенсує втрату якості;

- тестове покриття (*test coverage*) – метрики «test coverage» вказують ступінь повноти різних типів тестування;

- надійність (*reliability*) – метрика, що оцінює ймовірність роботи системи без відмов. Дана метрика може бути отримана в рамках традиційного підходу;

- профілі помилок (*fault profiles*) – метрика, измеряющая кумулятивне число виявлених помилок;

- ступінь задоволення потреб замовника (*customer satisfaction*) – метрика, що оцінює ступінь відповідності програмного забезпечення очікуванням та вимогам замовника. Дана метрика може бути оцінена перед поставкою на етапі дослідної експлуатації на основі прогнозуючих параметрів.

7.1 Оцінювання якості показників керування

Розглянемо більш доцільно три фундаментальні набори метрик керування: робота та прогрес, фінансовий стан і прогрес у наймі персоналу. Вивчаючи ці аспекти, менеджери проекту можуть оцінити, чи вкладається проект у рамки бюджету та термінів.

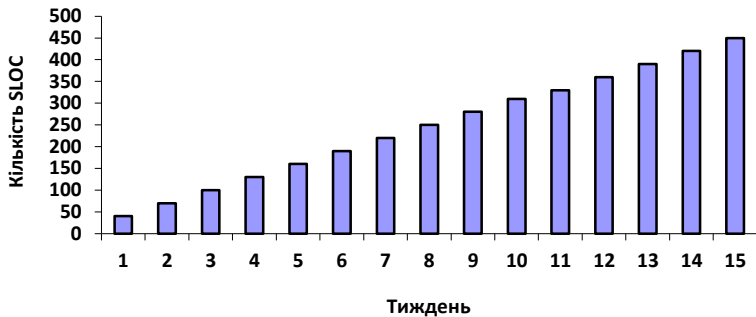
7.1.1 Оцінювання якості роботи та прогресу

Різні види діяльності при ітераційному розробленні проекту можуть бути виміряні за допомогою визначення обсягу запланованої роботи, виражених в об'єктивних одиницях виміру, з подальшим відстеженням прогресу (обсяг роботи, що виконана на цей момент) по відношенню до плану. У кожній великій команді в організації повинен існувати принаймні один аспект прогресу, щодо якого проводяться вимірювання.

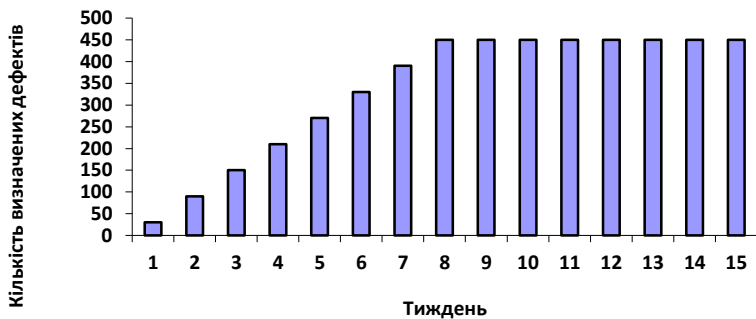
Наприклад, для архітекторів це кількість розглянутих варіантів використання, для розробників – кількість рядків коду (*SLOC*) (рис. 7.1), для тестувальників – кількість відкритих *SCO*, час (в

годинах) виконаного тестування та відповідність критеріям оцінки, для менеджера проєкту – кількість пройдених контрольних точок.

На рис. 7.1 а зображені кількість розроблених *SLOC* за визначений період часу. На рис. 7.1 б – кількість визначених дефектів. Оскільки ці показники залежать від етапу розробки, то для подальшого аналізу рекомендується використовувати методи експоненціального згладжування.



а)



б)

Рис.7.1.Очікуваний прогрес типового проєкту з трьома основними версіями, а) кількість *SLOC*, б) кількість дефектів

Також показником керування можуть слугувати функціональні точки (*Function Points, FP*) та їх похідні і різновиди, які обчислюються на базі не вихідного коду, а вимог користувачів, специфікацій, описів прецедентів (наприклад, точки прецедентів, *Use Case Points, UCP*) та ін. Відповідно до назви, функціональними точками вважається ряд функцій, що розробляються на основі специфікації вимог. У більшості випадків головне призначення цієї групи метрик, незалежно від способу їх обчислення, полягає в тому, щоб оцінити обсяг робіт за проектом, і, відповідно, бути основою для таких показників, як вартість і тривалість його реалізації.

7.1.2 Оцінювання фінансового стану проекту

Для контролю над проектом потрібно постійно обчислювати витрати протягом всього життєвого циклу. Використання метрик роботи та прогресу може дати більш об'єктивну оцінку технічного прогресу для її порівняння з фінансовими витратами. Для ітераційного процесу розроблення є важливим детально планувати найближчі дії (зазвичай для тимчасового інтервалу менше шести місяців) і складати плани подальших дій у вигляді грубих оцінок, які повинні уточнюватися по мірі того, як минає інтервал часу поточної ітерації і стає критичним планування наступної ітерації.

Відстеження фінансового прогресу зазвичай приймає форму, характерну для конкретної організації. Єдиним загальним підходом до вимірювання фінансового стану є застосування системи придбаної вартості, що дозволяє детально визначити витрати часу. Його недоліком при використанні для проектів по створенню ПЗ традиційно є неможливість об'єктивної і точної оцінки технічного прогресу (відсотка завершеної роботи). Це властиво стадії проектування системи проекту. Проте системи придбаної вартості довели свою ефективність на стадії виробництва, для якої характерні правильне відстеження реальних показників порівняно з планами та передбачувані результати. Інші метрики забезпечують основу для отримання докладних і реалістичних кількісних даних по зворотному зв'язку для планування та звірки, особливо на стадії виробництва, де фінансові та часові витрати є найвищими.

Сучасні процеси створення ПЗ легко піддаються вимірюванню фінансового стану за допомогою підходу придбаної вартості. Основними параметрами системи, які звичайно обчислюються в грошових одиницях, є наступні:

- план вартості: вид кривої планованих для проекту витрат щодо планованих термінів. Для більшості проектів зі створення ПЗ (та інших трудомістких проектів) вид цієї кривої повторює вигляд кривої найму персоналу;

- реальний прогрес: технічне виконання щодо планованого прогресу, який лежить в основі кривої фінансових витрат. У проекті, який нормально розвивається, реальний прогрес невідступно слідує за запланованим прогресом;

- реальна вартість: крива реальних витрат на проект для реального графіка виконання. У проекті, що нормально розвивається, ця крива невідступно слідує за планованою кривою;

- придбана вартість: визначається вартістю реального прогресу;

- неузгодженість витрат: різниця між реальними витратами та набутою вартістю. Позитивні значення відносяться до ситуацій, пов'язаних з перевитратою бюджету; негативні значення відповідають ситуаціям, пов'язаним з економією бюджету;

- неузгодженість термінів: різниця між запланованими витратами і придбаною вартістю. Позитивні значення відносяться до ситуацій, пов'язаних з відставанням від графіка; негативні значення відповідають ситуаціям, пов'язаним з випередженням графіка.

На рис. 7.2 надається графічна уява цих параметрів і приводиться простий приклад стану проекту.

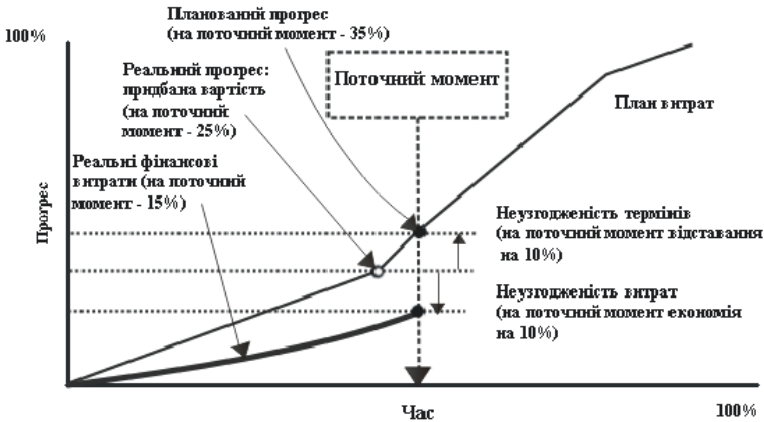


Рис. 7.2. Основні параметри системи придбаної вартості

Основною метою інших основних метрик є забезпечення команд з керування та розроблення більш об'єктивним підходом до оцінки реального прогресу з найбільшою точністю. З усіх параметрів системи придбаної вартості реальний прогрес є найбільш суб'єктивною оцінкою. Оскільки більшість менеджерів точно знають, скільки витрат вони виробили і яка частина часу використана, то розбіжності при виконанні точних оцінок фінансового благополуччя стосуються правильності оцінки реального прогресу.

7.1.3 Оцінювання найму персоналу

Ітераційна розробка повинна спочатку вестися невеликою командою до тих пір, поки всі ризики у вимогах і архітектурі не будуть дозволені відповідним чином. Залежно від суміщення ітерацій та інших характерних для даного проекту обставин, штатний розклад може змінюватися. Типова для виконання окремих одноразових завдань з розроблення (наприклад, для побудови корпоративної інформаційної системи) зміна чисельності задіяних у проєкті співробітників наведена на рис. 7.3. Розумно очікувати, що група супроводу для такого роду проєктів виявиться менше групи

розроблення. При створенні комерційного продукту розміри груп підтримки та розроблення можуть виявитися однаковими. Коли довгоживучі продукти, що постійно піддаються змінам в рамках постійного поліпшення якості, супровід є всього лише постійним створенням нових поліпшених версій.

Початкова стадія	Прояснення	Конструювання	Введення до дії
Робота: 5% Терміни: 10%	Робота: 20% Терміни: 30%	Робота: 65% Терміни: 50%	Робота: 10% Терміни: 10%

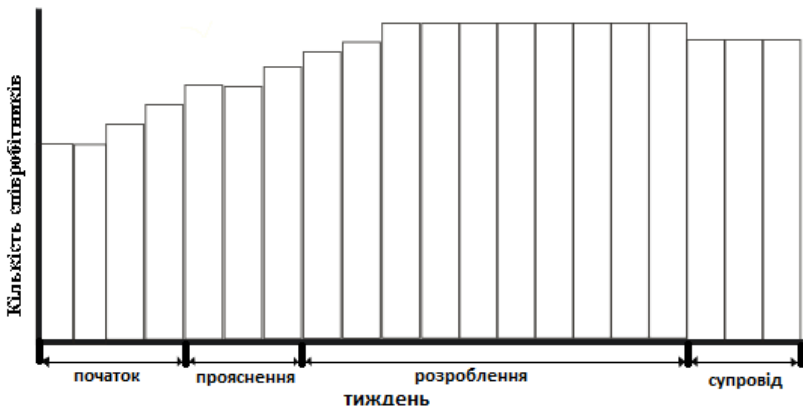


Рис.7.3. Типова зміна чисельності співробітників

Відстеження реального найму співробітників у порівнянні з запланованим є необхідним параметром керування, що легко розуміється. Існує ще один важливий для керування показник зміни рушійної сили проекту – відношення співробітників, що звільнилися, до знову прийнятих. Збільшення штату може уповільнити загальний прогрес проекту, оскільки постійно працюючим співробітникам доведеться витрачати продуктивний час на введення нових співробітників до курсу справи. Низький відсоток звільнення хороших

співробітників є ознакою успіху. Сильною мотивацією для розробників є можливість привести щось у дію; це постійна тема, що лежить в основі ефективного ітераційного процесу розроблення. Якщо така мотивація відсутня, хороші розробники йдуть до іншого джерела. Зростання незапланованого звільнення людей – тобто кількості людей, які передчасно залишають проект, – є одним з найбільш наочних показників того, що цей проект чекають проблеми. Причини передчасного звільнення можуть бути різними, але зазвичай це незадоволеність персоналу методами керування, відсутність командної роботи або висока ймовірність провалу в процесі досягнення поставлених цілей.

7.2 Оцінювання показників якості розробки

Чотири показника якості засновані насамперед на вимірі змін у ПЗ при зміні базових даних, що мають відношення до розроблення (таких, як проектні моделі та вихідний код).

7.2.1 Оцінювання інтенсивності змін і стабільності

Загальна інтенсивність змін – це особливий показник прогресу та якості. Інтенсивність змін обчислюється як кількість запитів на внесення змін до ПЗ, відкритих і закритих протягом усього життєвого циклу (рис. 7.4). Цей параметр може визначатися в залежності від типу внесених змін у розрахунок на одну версію, на всі версії, на одну команду, на один компонент, на одну підсистему і т.д. Розглянутий спільно з метриками роботи та прогресу, він дозволяє оцінити стабільність ПЗ і його рух у бік стабільності (чи нестабільності). Стабільність визначається як відношення між відкритими та закритими SCO. Інтенсивність змін, віднесена до графіка версії, дає уявлення про передбачуваність графіка робіт, що є основною цінністю даної метрики та показником того, наскільки добре йде процес.

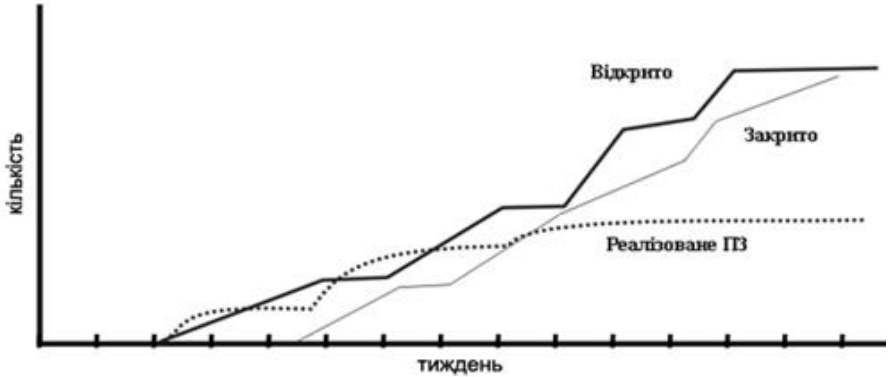


Рис. 7.4. Очікувана стабільність протягом життєвого циклу благополучного проекту

Наступні три метрики якості мають більше відношення до якості продукту.

7.2.2 Дефекти та коефіцієнт дефективності

Дефективність визначається як середня міра змін, що являє собою обсяг базового ПЗ, що вимагає доробки (може виражатися в *SLOC*, функціональних точках, компонентах, підсистемах, файлах і т.п.). Коефіцієнт дефективності – це тенденція зміни середньої кількості дефектів зі зміною часу. Для успішного проекту очікувана тенденція – зменшення або стабільність (рис. 7.5).

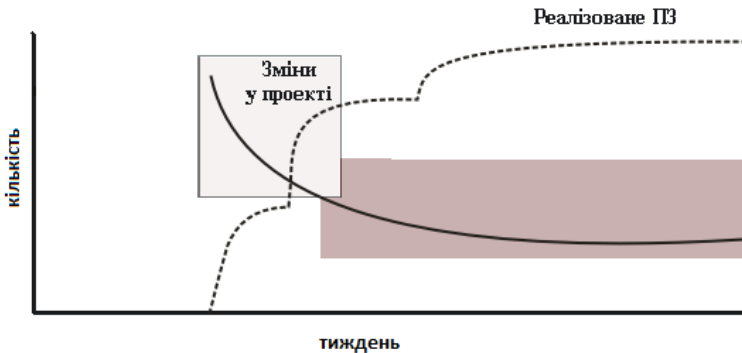


Рис. 7.5. Очікуваний коефіцієнт дефективності протягом життєвого циклу благополучного проекту

Цей показник дозволяє зрозуміти, який характер носять зміни ПЗ – доброякісний або злоякісний. У разі зрілого процесу ітераційного розроблення зміни на ранніх етапах зазвичай призводять до більшого обсягу відбракування, ніж на пізніших. Тенденція до збільшення дефектів з часом ясно вказує на те, що супровідність продукту викликає підозри.

7.2.3 Оцінювання вартості доробки і адаптованості

Доробка визначається як середня вартість внесення змін, до якої відносять витрати на аналіз, ухвалення рішення, повторне тестування всіх змін в основі ПЗ. Адаптованість визначається як тенденція зміни кількості доробок в залежності від часу. Для якісного проекту очікувана тенденція – зменшення або стабільність (рис.7.6).

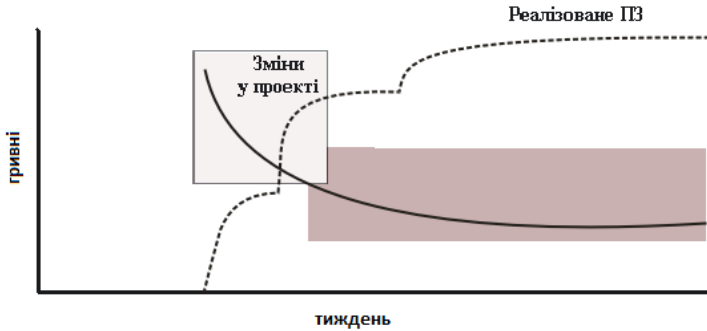


Рис. 7.6. Очікувана адаптованість протягом життєвого циклу благополучного проекту

Не всі зміни однакові. Деякі з них можуть бути внесені за одну людино-годину, інші вимагають декількох людино-тижнів. Ця метрика дозволяє вимірювати обсяг доробок. У сформованому ітераційному процесі зміни на ранніх етапах (зміни в архітектурі, які впливають на велику кількість компонентів і людей), як правило, вимагають великих доробок, ніж на пізніх (зміни при розробці, які зазвичай обмежуються одним компонентом або людиною). Тенденції до збільшення доробок з часом ясно вказують на те, що супровід продукту викликає підозри.

7.2.4 Оцінювання середнього часу до відмови

MTBF – це середній час використання ПЗ між двома відмовами. У загальному випадку *MTBF* можна обчислити, поділивши час тестування на кількість *SCO* типів 0 і 1. Завершеність визначається як тенденція зміни *MTBF* з плином часу (рис. 7.7).

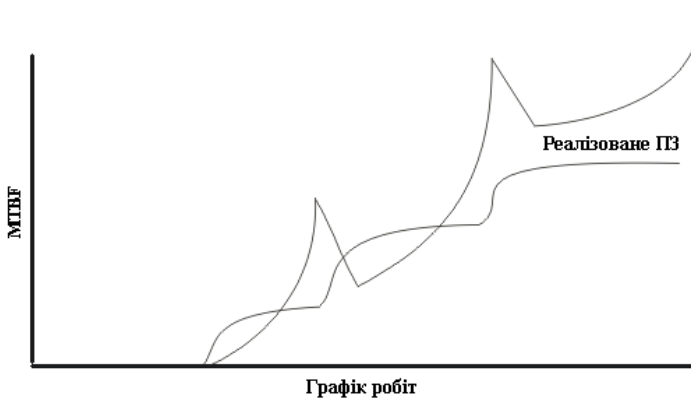


Рис. 7.7. Очікувана завершеність протягом життєвого циклу благополучного проекту

Раннє визначення завершеності вимагає створення ефективної інфраструктури для тестування. Традиційні підходи до тестування монолітних комп'ютерних програм приділяють основну увагу досягненню повного тестового покриття кожного рядка коду, кожної гілки програми і т.д. У сучасних розподілених і розбитих на компоненти програмних системах таке вичерпне тестове покриття можливо досягти тільки для окремих компонентів. Системи компонентів більш ефективно тестуються з використанням статистичних методів. Відповідно параметр «завершеність» визначає статистику за час використання, а не охоплює весь продукт.

Помилки в ПЗ можуть бути розбиті на дві категорії: детерміновані та недетерміновані. Перші – це клас помилок, які завжди виявляються, коли ПЗ використовується певним чином. Такі помилки в більшості випадків є наслідком помилок кодування, а необхідні зміни зазвичай обмежуються одним компонентом. Помилки Інший клас – це відмови ПЗ, що відбуваються випадково з деякою ймовірністю при настанні певної ситуації. Ці помилки майже завжди виникають через помилки, допущені при проектуванні (і часто потребують внесення змін до безлічі компонентів), і звичайно не відтворюються навіть у тих випадках, коли ПЗ використовується одним і тим же способом. Для забезпечення адекватного тестового покриття та виправлення статистично значущих помилок другого

класу потрібно екстенсивне статистичне тестування за реальними та випадковими сценаріями використання.

Таблиця 7.1 – Стандартні зразки зміни метрик протягом життєвого циклу для RUP

	Початок	Проектуван- ня	Розробка	Впровад- ження
Прогрес Архітектура Програми	5% 30% <5%	25% 90% 20%	90% 100% 85%	100% 100% 100%
Фінальний стан Робота Час	Низькі 5% 10%	Помірні 25% 40%	Високі 90% 90%	Високі 100% 100%
Динаміка у наймі персоналу	Невелика команда	Зростаюче	Стале	Змінне
Стабільність Архітектура Програми	Змінна Змінна Змінна	Помірна Помірна Змінна	Помірна Стабільна Помірна	Висока Стабільна Стабільна
Коефіцієнт дефективності Архітектура Програми	50% – 100% >50% >80%	25% – 50% >50% >80%	<25% <15% <25%	5% – 10% <5% <10%
Адаптованість Архітектура Програми	Різна Різна Різна	Різна Помірна Різна	Якісна Якісна Помірна	Якісна Якісна Якісна
Завершеність Архітектура Програми	Прототип Прототип Прототип	Незначна Застосовна Незначна	Застосовна Повна Застосовна	Повна Повна Повна

Традиційні програмні продукти, в яких єдина програма виконується на єдиному процесорі, як правило, містять тільки помилки 1 класу. Сучасні розподілені системи з безліччю взаємодіючих компонентів, що виконуються в мережі процесорів, виявляються уразливими до помилок 2 класу, які складно виявляти, аналізувати та виправляти. Кращий спосіб зробити програмний продукт досконалим полягає у створенні з самого початку такої інфраструктури тестування, яка допускає виконання випадкових сценаріїв використання на ранніх стадіях життєвого циклу і постійно

збільшує ширину і глибину сценаріїв для охоплення критичних по надійності компонентів.

По мірі свого створення базові версії ПЗ повинні постійно перевірятися за допомогою сценаріїв тестування. На основі цього тестування можна визначити параметри надійності. Осмислена оцінка завершеності продукту може бути виконана за рахунок максимізації часу тестування (з використанням незалежного середовища тестування, автоматизованих регресійних тестів, випадкового статистичного тестування, тестування в години, які слідують за великими навантаженнями, і т.д.). Такий підхід надає потужний механізм для автоматизації тестування на самих ранніх стадіях життєвого циклу. Цей метод може бути застосований також для моніторингу продуктивності та вимірювання надійності.

7.3 Використання метрик ПЗ

Будь які вимірювання корисні, але висновки робить особа приймаюча рішення (Менеджер проекту, керівник, команди). Вимірювання тільки дають дані, які допомагають правильно ставити питання, розуміти контекст і приймати об'єктивні рішення. Оскільки природа проектів зі створення ПЗ надзвичайно динамічна, то можливість виконувати такі вимірювання повинна існувати в будь-який час і бути застосовна до різних частин мінливого продукту (версія, компонент, клас). Виміри повинні проводитися таким чином, щоб можна було оцінити тенденції замість показників з часом. На практиці така ситуація досяжна тільки в тих проектах, де підтримується доступ в онлайн-режимі до метрик, що автоматично визначаються, як до побічного продукту середовища розроблення/інтеграції.

У таблиці 7.2 зведені розглянуті метрики. Кожна з них має два виміри: статичне значення, що використовується в якості мети, і динамічна тенденція, що застосовується для досягнення мети. Значення метрик дають уявлення тільки в одному вимірі; з точки зору керування процесом важливішими виявляються тенденції їх зміни. Тенденції зміни метрик у часі дозволяють зрозуміти, в яких напрямках змінюються продукт і процес. Ітераційному процесу притаманне

внесення змін, а вимір цих змін є важливим аспектом параметрів програми. Абсолютні значення продуктивності і якісних поліпшень вторинні до тих пір, поки не буде вирішена фундаментальна задача керування: передбачуваність витрат коштів і часу для заданого рівня якості.

Таблиця 7.2 – Сім основних метрик керування проектами

Метрика	Мета	Способи визначення
Робота і прогрес	Планування ітерацій, порівняння плану і реальних досягнень, показник керування	<i>SLOC</i> , функціональні точки, об'єктні точки, сценарії, варіанти тестування, <i>SCO</i>
Фінансовий стан	Розуміння фінансових питань, порівняння плану і реальних значень, показник керування	Щомісячні витрати, кількість використовуваних протягом місяця співробітників з повним робочим днем, відсоток витраченого бюджету
Динаміка найму персоналу	План споживання ресурсів у порівнянні з реальністю, рівень найму, рівень скорочення чисельності персоналу	Кількість нових співробітників за місяць, кількість що пішли співробітників за місяць
Інтенсивність змін і стабільність	Планування ітерацій, показник дотримання графіка	Кількість поданих <i>SCO</i> в порівнянні з кількістю закритих <i>SCO</i> за типами (0,1,2,3,4), що припадають на одну версію/компонент/підсистему
Дефекти і коефіцієнт дефективності	Досягнення мети, доробка ПЗ, показник якості	Кількість перероблених <i>SLOC</i> на одну зміну за типами (0,1,2,3,4), що припадають на одну версію/компонент/ підсистему
Доробки і адаптованість	Досягнення мети, переробка ПЗ, показник якості	Середня кількість годин, витрачених на одну зміну за типами (0,1,2,3,4), що припадають на одну версію/компонент/підсистему
<i>MTBF</i> і завершеність	Покриття/ адекватність тестування, простота використання, показник якості	Число відмов, кількість годин між відмовами при тестуванні, що припадають на одну версію/компонент/підсистему

Після того як сімейство метрик визначено, вони можуть бути використані проектом або організацією для передбачення витрат, термінів і якості виконання різних робіт у майбутньому.

Коли метрики вказують на деяку проблему, важливо вникнути в усі симптоми, щоб поставити діагноз. Метрики зазвичай відображають ефекти; для розуміння причин потрібен синтез багатьох

точок зору та висновків. Наприклад, умови, необхідні для правильної інтерпретації таких ситуацій:

- мала кількість запитів на внесення змін може означати, що ПЗ досконало і вільно від помилок, а може означати, що команда, відповідальна за тестування, перебуває у відпустці;

- запит на внесення змін, що залишається відкритим протягом тривалого часу, може означати, що помилка була легко діагностована, але її виправлення потребувало значних доробок, або може означати, що її діагностика зайняла дуже багато часу, а виправлення полягало у внесенні змін в один єдиний рядок коду;

- велике зростання персоналу в поточному місяці може призвести до пропорційного збільшення прогресу, якщо новий персонал складається з навчених людей, які відразу ж приступлять до продуктивної праці. І він же може викликати уповільнення прогресу, якщо найняті ненавчені люди, яким потрібно найрізноманітніша підтримка з боку тих, хто зайнятий продуктивною працею, для введення в курс справи;

- поточні значення та поточні тенденції зміни метрик використовуються в якості параметрів керування проектом;

- реальні значення наведених метрик можуть змінюватися в широких межах в рамках проектів, організацій та областей застосування. Однак відносні тенденції зміни протягом стадій проекту повинні слідувати загальним прикладам, наведеним у таблиці 7.1. Стабільна організація-розробник повинна вміти описувати ті значення метрик, до яких вона прагнучиме, що є більш інформативним і точним відносно її основної діяльності і конкретних процесів.

Якісні оцінки не можуть виконуватися самими метриками, для цього потрібні інтелектуальні зусилля з боку, наприклад, менеджерів проектів по створенню ПЗ.

7.4 Практичні завдання

Завдання 1. Обґрунтуйте та оберіть для проекту метрики, спираючись на які, можливо отримати найбільш важливу інформацію по проекту. Порівняйте ваші обрані метрики з метриками в таблиці 7.2.

Завдання 2. Для наступних метрик наведіть шкалу для трьох положень (поточний, гірший і плановий):

- робота і прогрес;
- фінансовий стан;
- динаміка найму персоналу;
- інтенсивність змін і стабільність;
- дефекти і коефіцієнт дефективності;
- доробки і адаптованість;
- *MTBF* і завершеність.

Завдання 3. Ви керівник проекту «Освіта on-line». Розплануйте розподіл трудомісткості етапів «початок», «проектування», «розробка», «впровадження» та оберіть метрики для відстежування відхилень від плану.

7.5 Контрольні запитання

1. Які метрики відносяться до показників якості керування?
2. Які метрики відносяться до показників якості розробки?
3. Наведіть основні параметри системи придбаної вартості.
4. Як можна інтерпретувати зростання об'ємів робіт?
5. Які тенденції є показниками оцінки найму персоналу?
6. Які кількісні метрики можуть використовуватись при розрахунку дефективності?
7. Що означає метрика «адаптованість»?
8. Які типи помилок зустрічаються при розробці ПЗ?

7.6 Література до розділу

Метрики якості програмних проектів: [9-11],[16],[42],[53-74],[117].

8 КЕРУВАННЯ КОНФІГУРАЦІЄЮ ПОГРАМНИХ ЗАСОБІВ

8.1 Основні визначення

Під конфігурацією розуміється конкретна версія програмної системи, що містить у собі функції, об'єднані між собою процедурами, зв'язками та параметрами, що задають режими функціонування системи.

Версія або конфігурація системи складається з:

- 1) базису конфігурації, формально створеної основи системи з окремо створених компонентів і документації, що дозволяють проводити подальший розвиток системи;
- 2) елементів конфігурації, виділених для керування або оброблення функції системи;
- 3) програмних компонентів, що виконують завдання сформованої версії системи.

Базис – конфігураційні певні технічні рішення, перелік головних елементів конфігурації, значень параметрів і спеціалізовані процедурні зв'язки та розгортання версії системи з компонентів заданої послідовності.

Чим більше в системі компонентів, тим більша ймовірність того, що деякі з них будуть містити помилки.

Керування конфігурацією (КК) – це процес забезпечення ідентифікації елементів конфігурації системи при її створенні для проведення систематичного контролю, обліку й аудиту внесених змін, а також для перевірки цілісності та працездатності системи.

Відповідно до стандарту *IEEE* 828-2005, керування конфігурацією містить:

- 1) ідентифікацію конфігурації;
- 2) контроль конфігурації;
- 3) облік статусу конфігурації.

Керування конфігурацією для великих систем створюється за допомогою методів і засобів, що забезпечують:

- ідентифікацію елементів системи;

- контроль внесених змін;
- можливість визначення фактичного стану системи при розробленні й експлуатації в будь-який момент часу.

Для досягнення цілей керування конфігурацією повинне проводитися планування й виконання проекту з урахуванням виникаючих обмежень операційних систем і обладнання замовників.

Процес змін містить у собі: визначення типів змін; організацію їх проведення; формування концепції допуску відхилень і відмов стосовно вимог проекту системи.

Результат внесених змін – це нова версія системи, документація з проведення випробувань і документація користувача на систему.

При внесенні змін проводиться контроль поточної версії системи з використанням репозитарію (*Rational Clear Case Source* і *Source Safe of Microsoft* – інструменти перевірки змін) і перевірка вихідного коду отриманої версії.

Планування конфігурації залежить від типу проекту, організаційних заходів, обмежень і загальних рекомендацій до керівництва конфігурацією.

До планування керування конфігурацією відносять:

- ідентифікацію;
- визначення статусу й аудиту конфігурації;
- керування змінами конфігурації.

До засобів забезпечення планування належать:

- система керування кодами, переведення й об'єднання компонентів у конфігураційні системи;
- базові бібліотеки та ресурси;
- спеціальні групи контролю системи та її конфігурацій;
- системи керування для ведення проекту та зберігання змін.

Ідентифікація конфігурації – іменування всіх елементів системи на основі системи класифікації й кодування елементів, а також подання та ведення версій конфігурації з використанням належних до неї елементів. Вона виконується за допомогою методів структуризації, класифікації й іменування елементів системи. При цьому проводиться:

- 1) визначення стратегії ідентифікації для одержання врахованої версії системи;
- 2) іменування складених елементів, частин і всієї конфігурації системи;

- 3) встановлення співвідношення між кількістю завдань, що виконуються, та кількістю пунктів конфігурації;
- 4) ведення версії системи та її документування;
- 5) набір елементів у базисі конфігурації і його формальне позначення.

Основи ідентифікації становить конфігураційний базис, набір формально розглянутої й затвердженої документації як основи для подальшого розвитку або розроблення системи.

Версія системи містить елементи конфігурації та систему для одержувача.

Керування версіями полягає:

- 1) в інтеграції або композиції коректної остаточної версії системи з елементів конфігурації, реалізованих на етапах життєвого циклу, а також з урахуванням апаратних засобів і інструментів побудови системи;

- 2) у виборі інструментів побудови версії оцінки можливостей середовища та засобів автоматизації процесу побудови окремих версій з коректними конфігураціями ПО й даних;

- 3) керування варіантами версій, що включають сукупність готових ідентифікованих елементів системи та задовольняють задані вимоги замовника до варіанта.

Після одержання нової версії системи замовникові передаються версія, конфігурація, документація й інструменти керування версіями для самостійного внесення змін при супроводі системи.

Контроль конфігурації – це перевірка й керування змінами системи при формуванні версії й експлуатації.

Предметом проведення контролю конфігурації є:

- 1) зміни в базисі конфігурації й пов'язане з ними коректування конфігурації;

- 2) дефекти й відхилення конфігурації продукту щодо затвердженого базису.

Це здійснюється за допомогою формальних процедур ініціалізації, аналізу, прийняття й контролю, виконання управлінських рішень щодо зміни виявлених дефектів і відхилень у конфігурації.

Після того, як зацікавлені учасники проекту досягли взаєморозуміння, відповідні проектні документи вважаються затвердженими і не можуть довільно модифікуватися. Отже, будь-яка потреба в зміні повинна пройти процедури:

- 1) реєстрації додатка на зміни;
- 2) аналізу впливу запропонованої зміни на наявний доробок, обсяг, трудомісткість, графік і вартість робіт із проекту;
- 3) ухвалення рішення про зміну: задовольнити, прийняти або відмовити;
- 4) реалізація затвердженої зміни й верифікація.

Другою складовою контролю конфігурації є керування невідповідностями між конфігурацією й елементами продукту та базисами конфігурації.

З погляду керування всі невідповідності прийнято ділити на дефекти й відхилення.

До дефектів відносять невідповідності, що мають відношення до цільового використання продукту за його призначенням. Все інше відносять до відхилень.

Якщо дефекти в продуктах носять негативний характер, то вони підлягають усуненню.

Облік статусу полягає в реєстрації та наданні інформації для ефективного керування конфігурацією. Предмет такого обліку – інформація про поточний статус ідентифікованих об'єктів, запропонованих змінах, а також про виявлені дефекти та відхилення.

Звіт про статус конфігурації є ключовим для прийняття управлінських рішень щодо проекту.

Дані обліку статусу конфігурації – вихідні дані для формування кількісних оцінок або метрик продуктивності.

Аудит конфігурації містить:

- 1) функціональний аудит конфігурації, що проводиться для підтвердження відповідності фактичних характеристик конфігурації продукту вимогам замовника;
- 2) фізичний аудит, що є підтвердженням взаємної відповідності документації й фактичної конфігурації продукту.

8.2 Функціонування систем керування конфігурацією

Системи управління конфігураціями можуть стати потужним засобом поліпшення взаємодії, продуктивності та якості шляхом автоматизації процесів та інтеграції інструментів, що

використовуються в даний час більшістю груп інженерів. Управління конфігураціями можна назвати мостом, що зв'язує групи розробників в компанії. Застосування системи управління конфігураціями дозволяє організувати свої зусилля по розробці, в основі яких лежать чіткі і повторювані процеси; при цьому, допомагаючи своїй групі в розстановці пріоритетів та управлінні життєвим циклом розробки, ви більшою мірою здатні забезпечити відповідність вимогам своїх клієнтів.

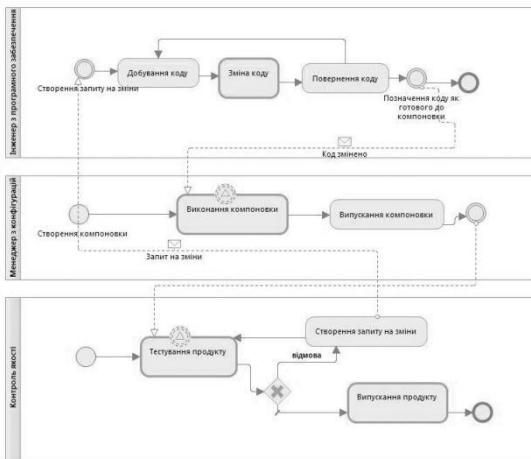


Рис. 8.1. Приклад процесів стандартної системи керування конфігурацією

При переході від проекту в чистому вигляді до того, що вимагає доставки користувачам, потрібно зібрати весь код разом. Після проведення тестування та інтеграції ваш клієнт випускає створений вами продукт.

Компоновки або збірки призначені головним чином для відділів розробки і тестування і можуть також використовуватися для пошуку помилок шляхом тестування та налагодження.

Після того як продукт був зібраний, групи тестування і розробки відразу кваліфікують продукт як готовий до випуску.

Якщо у вас розробка виконується кількома підрозділами, можна налаштувати час зборки таким чином, щоб виконувати кілька збірок

за 24 години. З іншого боку, якщо збірка займає мало часу, можна виконувати її кілька разів в день.

Випуски продуктів залежать від успішності тестування компоновки та від вимог клієнтів.

Є кілька причин для автоматизації компоновки. Перша причина - узгодженість. Якщо не збирати продукт однаковим способом кожен раз, як дізнатися, відбулася реальна помилка або помилкова помилка? Ніщо так не сповільнює цикл розробки, як неузгодженість процесу складання.

Друга причина - повторюваність. Дуже часто буває так, що група розробників вже працює над наступним випуском, тоді як клієнт знаходить помилки в попередньому випуску. У таких випадках зміни в код потрібно внести в попередньому випуску, і ці попередні випуски потрібно повторно зібрати і випустити. Якщо збірка не автоматизована, цей процес може бути дуже виснажливим і схильним до помилок.

І нарешті, потрібна можливість поширення зборок по декількох ресурсів в мережі обчислень (computation grid) або мережі даних (data grid). Це дозволить скоротити час циклу складання та тестування та підвищити якість і продуктивність вашої групи.

Так як зазвичай група розробників складається не з однієї людини, зв'язок між учасниками групи дуже важлива. Існує два основних види зв'язків, які потрібно враховувати при розробці системи збирання і випуску: негайне повідомлення і накопичення інформації.

При негайному повідомленні інженер з компоновки або інженер по випуску оповіщається про виникнення проблем із збіркою або про її успішному виконанні. Для цього краще всього використовувати електронну пошту.

При накопиченні інформації, вонна повинна містити повні відомості про виконану збірку. Кількість інформації про збірку, повністю залежить від вашої організації. Зазвичай для такого роду зв'язку використовується веб-сайт. Можна накопичувати такі відомості як: попередження та помилки компіляції; попередження та помилки очищення; результати тестування: пройдено/ не пройдено; результати інтерпретації (lint); файли журналів для різних етапів процесу; інформація про час; платформи та комп'ютери.

8.3 Практичні завдання

Завдання 1. Ознайомтеся з системою контролю версій проекту *IBM Rational Clear Case*.

1.1 Створіть VOB (*Version Object Base*) проекту *Microsoft Visual Studio*.

1.2 Створити View проекту.

1.3 Поставити проект під конфігураційне керування.

Завдання 2. Ознайомтеся з розподіленою системою керування версіями файлів Git та Mercuria, порівняйте функціональні можливості цих систем, визначте їх переваги та недоліки.

8.4 Контрольні запитання

1. Які види робіт містить керування конфігурацією? Наведіть їх характеристики.

2. В чому полягає керування версіями?

3. Назвіть складові контролю конфігурації.

8.3 Література до розділу

Керування конфігурацією:[2],[115].

ПЕРЕЛІК ПОСИЛАНЬ

1. Arras, P. E-learning concept for the properties of materials remote study/ Arras, P.; Tabunshchyk, G.; Kozik, T.// IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2013 Volume: 02, 2013, P3: 742–747. DOI: 10.1109/IDAACS.2013.6663024
2. Bellagio, D. E. Software Configuration Management Strategies and IBM Rational ClearCase: A Practical Introduction (2nd Edition) // D. E. Bellagio, T. J. Milligan. Publisher: IBM Press, - 2005. -384 p
3. Bergmann, S. Real-World Solutions for Developing High-Quality PHP Frameworks and Applications / S. Bergmann, S. Pribsch. – John Wiley & Sons, 2011 – 480 p.
4. Arras, P. E-learning environment for the remote study in material properties courses/ Arras, P., Kolot, Y., Galyna, T., Kozik, T // International Journal of Computing, 12 (3), 2014, 233–238.
5. Arras, P. Architectural Characteristics and Educational Possibilities of the Remote Laboratory in Materials Properties/ P. Arras, G. Tabunshchyk, Ye. Kolot, B. Tanghe// REV2014 Conference26–28 February 2014, Porto, Portugal, PP. 94–97
6. Bevan, N. Specifying and Measuring Quality in Use. – Proceedings of the 22nd international conference on Software engineering, 2000. – PP. 322–331.
7. Boehm, B.W. Characteristics of Software Quality Text. / B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. MacLeod, M. J. Merritt. North Holland, 1978. – 524 p.
8. Boehm, B.W. Software Risk Management. – IEEE Computer Society Press, CA, 1989. – 496 p.
9. Bragina, T. A Modified Method for Estimating Software Projects Labor Costs / T. Bragina, G. Tabunshchik // TCSET'2012: Proc. Of XI Int. Conf. Modern Problems of Radio Engineering, Telecommunications and Computer Science. – Lviv: Lviv Polytechnic National University, 2012. – P. 245.
10. Bragina, T. Comparative Analysis of Software Development Models for Electro-technical Systems / T. Bragina, G. Tabunshchik // Telecommunications and Computer Science TCSET'2010: Proc. of Int.

Conf. on Modern Problem of Radio Engineering. Lviv–Slavsko, 2010. – P. 347.

11. Bragina, T. Fuzzy model for the software projects design risk analysis /T. Bragina, G.Tabunshchik // CADSM 2011: Proc. Of XI Int. Conf. The Experience of Designing and Application of CAD Systems in Microelectronics. – Lviv: Lviv Polytechnic National University, 2011. – C. 335-341.

12. Capability Maturity Model for Software, Version 1.1 (CMU/SEI-93-TR-24) / Paulk, Mark C., and others, Software Engineering Institute, Carnegie Mellon University. – Pittsburgh, Pa.: SEI,CMU, 1993. – 88 p.

13. Arras, P. Iterative Pattern for the Embedding of Remote Laboratories in the Educational Process / P. Arras, K. Henke, G. Tabunshchik, D. V. Merode // 12th International Conference on Remote Engineering and Virtual Instrumentation (REV 2015) 25–28 February 2015, Bangkok, Thailand, PP. 52–55 (DOI 978–1–4799–7838–0/15)

14. Cook, C. R. Information Theory Metric for Assembly Language. – Software Engineering Strategies, 1993. – PP.52-60.

15. Dromey, G. Cornering the Chimera. – IEEE Software, January, 1996. – PP. 33-43.

16. Henke, K. Using Interactive Hybrid Online Labs for Rapid Prototyping of Digital Systems/ G. Tabunshchik, H-D Wuttke, St. Ostendorff, T. Vietzke //REV2014 Conference26–28 February 2014, Porto, Portugal, PP.48–59

17. Grady, R. Software Metrics: Establishing a Company-wide Program / R. Grady, D. Caswell. – Prentice Hall, 1987. – 275 p.

18. Henke, K. Using Interactive Hybrid Online Labs for Rapid Prototyping of Digital Systems/G. Tabunshchik, H-D Wuttke, St. Ostendorff, T. Vietzke // iJOE, Volume 10, Issue 5, 2014, – PP. 57–62.

19. Kozik, T. Techniques and tools for virtual and remote experiments/ Kozik T., Arras P., Tabunshchik G. // Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій: зб. тез доп. VII міжнар. наук. – практ. конф. (Запоріжжя, 17–19 вересня 2014 р.). – Запоріжжя: ЗНТУ, 2014. – С. 112–113.

20. «JTAG Boundary – Тестування сканування в MAX V пристроїв» <http://www.altera.com/literature/hb/max-v/mv51008.pdf>.

21. IEEE Standard for a Software Quality Metrics Methodology Copyright ©, Institute of Electrical and Electronics Engineers, Inc. 2005.

22. IEEE/EIA 12207.0-1996 Industry Implementation of International Standard ISO/IEC 12207:1995 – New York, Mar. 1998.

23. International Standard ISO/IEC 9126. Information technology – Software product evaluation – Quality characteristics and guidelines for their use. International Organization for Standardization, International Electrotechnical Commission, Geneva, 1991.

24. ISO 10007:2003 Quality management systems - Guidelines for configuration management: [Электрон. ресурс]. – http://www.iso.org/iso/catalogue_detail?csnumber=36644.

25. ISO 9001:2008: [Электрон. ресурс]. – Режим доступа: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=46486

26. ISO/IEC 14598-6:2001 Software engineering - Product evaluation - Part 6: Documentation of evaluation modules [Электрон. ресурс]. – Режим доступа: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=24907.

27. ISO/IEC 14764:2006 Software Engineering - Software Life Cycle Processes – Maintenance [Электрон. ресурс]. – http://www.iso.org/iso/catalogue_detail.htm?csnumber=39064

28. ISO/IEC 15504: [Электрон. ресурс]. – Режим доступа: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50519.

29. ISO/IEC 15910:1999 Информационные технологии. Процесс создания документации пользователя программными средствами [Электрон. ресурс]. – http://www.iso.org/iso/ru/home/store/catalogue_tc/catalogue_detail.htm?csnumber=29509.

30. ISO/IEC 18019:2004 Программное обеспечение и системотехника. Рекомендации по проектированию и подготовке документации пользователя по прикладному программному обеспечению [Электрон. ресурс]. – http://www.iso.org/iso/ru/home/store/catalogue_ics/catalogue_detail_ics.htm?ics1=35&ics2=080&ics3=&csnumber=30804.

31. ISO/IEC 25000:2005 Software Engineering - Software product Quality Requirements and Evaluation (SQuARE) - Guide to SQuARE [Электрон. ресурс]. – http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?ics1=35&ics2=80&ics3=&csnumber=35683.

32. ISO/IEC 90003:2004 Software engineering - Guidelines for the application of ISO 9001:2000 to computer software [Електрон. ресурс]. – http://www.iso.org/iso/catalogue_detail?csnumber=35867.

33. ISO/IEC 9126-1. 2001. Software engineering – Software product quality – Part 1: Quality model. Geneva, Switzerland: International Organization for Standardization.

34. ISO/IEC DTR 9126-3. 2000. Software engineering – Software product quality – Part 3: Internal metric. Geneva, Switzerland: International Organization for Standardization.

35. ISO/IEC TR 9294:2005 Информационные технологии. Руководящие указания по управлению документацией на программное обеспечение [Електрон. ресурс]. – http://www.iso.org/iso/ru/home/store/catalogue_tc/catalogue_detail.htm?csnumber=37460.

36. ISO/IEC DTR 19759 Software Engineering – Guide to the Software Engineering Body of Knowledge – SWEBOK. [Електрон. ресурс]. – Режим доступа: http://www.swebok.org/ironman/pdf/SWEBOK_Guide_2004.pdf.

37. ISO/IEC TR 12182:1998 Information technology – Categorization of software [Електрон. ресурс]. – Режим доступа: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=1389&commid=45086.

38. ISO/IEC TR 15846:1998 Информационные технологии. Процессы жизненного цикла программного обеспечения. Управление конфигурацией of software [Електрон. ресурс]. – Режим доступа: http://www.iso.org/iso/ru/home/store/catalogue_tc/catalogue_detail.htm?csnumber=30516.

39. Брагина, Т. И. Информационная технология риск-ориентированного оценивания функциональности web-ориентированных систем /Т.И. Брагина, Г. В. Табунщик // Системи обробки інформації. Вип.2 (118). – 2014. – С. 245–252.

40. Брагина, Т. И. Риск-ориентированный метод тестирования интегрированных баз данных /Т.И. Брагина, Г. В. Табунщик // «Электротехнические и компьютерные системы». – Одесса, 2014. – № 13 (89). – С. 223–230.

41. Притула А. В. Практично-орієнтовані методи викладання в галузі вбудованих систем/ Притула А. В., Пархоменко А. В., Табунщик Г.В // Сучасні проблеми і досягнення в галузі радіотехніки,

телекомунікацій та інформаційних технологій: зб. тез доп. VII міжнар. наук. – практ. конф. (Запоріжжя, 17–19 вересня 2014 р.). – Запоріжжя: ЗНТУ, 2014. – С. 216–217.

42. Liggesmeyer, P. Software-Qualität: Testen, Analysieren und Verifizieren von Software, 2. Auflage. – Spektrum Akademischer Verlag, 2009. – 526 p.

43. McCabe, T.J. A complexity measure. – IEEE Transactions on Software Engineering, vol. SE-2, no. 4, , December, 1976. – PP. 308-320.

44. McCall, J. Factors in Software Quality Text. / J. McCall, P. Richards, G. Walters: 3 vol. NTIS AD-A049-014, AD-A049-015, AD-A049-055, November 1977.

45. Pandian, C. R. Software Metrics: A Guide to Planning, Analysis, and Application. – Auerbach Publications, 2004. – 312 p.

46. Schulmeyer, G. G. Verification and Validation of Modern Software-Intensive Systems / G. G. Schulmeyer, G. R. Mackenzie. – NJ: Prentice Hall, 2000. – 493 p.

47. Шитикова Е. В. оптимизация процесса исследовательских испытаний сложных технических систем/ Шитиковае. В., Табунщик Г. В. // Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій: зб. тез доп. VII міжнар. наук. – практ. конф. (Запоріжжя, 17–19 вересня 2014 р.) . – Запоріжжя: ЗНТУ, 2014. – С. 252–253.

48. Software Process Improvement in Regions of Europe: [Електрон. ресурс]. – Режим доступа: <http://www.cse.dcu.ie/spire>

49. Tabunshchuk, G. Comparative Analysis of Software Development Models for Electro-technical Systems/ Tabunshchuk G., Bragina T. // Proc. of the Xth Int. Conf. TCSET'2010, Lviv-Slavske, Ukraine. Febr., -2010. - PP. 347

50. Щербак Н.В. Обучающий эксперимент для интерактивной образовательной среды /Щербак Н.В., Табунщик Г. В. //Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій: зб. тез доп. VII міжнар. наук. – практ. конф. (Запоріжжя, 17–19 вересня 2014 р.). – Запоріжжя: ЗНТУ, 2014. – С.254–255.

51. А.с. 46858, Україна. Комп'ютерна програма «Бібліотека методів тестування» / Т.І. Брагіна, Г.В. Табунщик; опубл.19.12.12.

52. Богданов, Д.В. Стандартизация жизненного цикла программных средств. – СПб.: Питер, 2000. – 210 с.

53. Брагіна, Т.І. Засоби аналізу ефективності архітектури інтернет-магазину / Т. І. Брагіна, Г.В. Табунщик // Системний аналіз та інформаційні технології: матеріали 12-ї Міжн. наук.-техн. конф., САІТ 2010, 25–29 травня, 2010, Київ, Україна. – С. 412.

54. Брагіна Т. И. Программная реализация методов оценки рисков программных проектов / Брагіна Т. И., Табунщик Г. В. // Тиждень науки – 2011: зб. тез доп. щоріч. наук.-практ. конф. викладачів, науковців, молодих учених, аспірантів, студентів ЗНТУ (Запоріжжя, 11–15 квіт. 2011 р.): в 4 т. Т. 2 / відп. ред. Ю. М. Внуков. – Запоріжжя: ЗНТУ, 2011. – С. 59-60.

55. Брагіна Т.И. Преобразование данных о рисках программных проектов для систем поддержки принятия решений / Брагіна Т.И., Табунщик Г.В. // Інформаційні технології та комп'ютерна інженерія: тези доповідей II Міжнар. наук.-практ. конф. – Х.: ХНЕУ, 2011.– С.15-16.

56. Брагіна, Т. И. Классификация моделей итеративной разработки программного обеспечения / Т. И. Брагіна Г. В. Табунщик // Системний аналіз. Інформатика. Управління: матеріали Всеукраїнської науково-практичної конференції, САІУ–2010. – Запоріжжя: КПУ, 2010. – С. 23–25.

57. Брагіна, Т. И. Комбинированный метод прогнозирования рисков / Брагіна Т. И., Табунщик Г. В. // 36. тез доп. VI міжнар. наук.-практ. конф. «Сучасні проблеми і досягнення в галузі радіотехніки, телеко-мунікацій та інформаційних технологій» (Запоріжжя, 19–21 вересня 2012 р.). – Запоріжжя: ЗНТУ, 2012. – С. 265-266.

58. Брагіна, Т. И. Методы управления неопределенностью в программных проектах с итеративным процессом разработки / Т.И. Брагіна, Г.В. Табунщик // Автоматика – 2010: зб. тез доповідей 17 міжн. конф. з автоматичного керування, 27-29 вересня, 2010, Харків, Україна. – Т. 1 – С. 101 – 102

59. Брагіна, Т. И. Методы управления рисками программных проектов / Т.И. Брагіна, Г.В. Табунщик // Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій: тези доповідей V Міжн. наук.-практ. конф., 22–24 вересня, 2010, Запоріжжя, Україна. – Т. 2 – С. 144 – 145.

60. Брагіна, Т. И. Метрики оценки качества web-систем / Брагіна Т. И., Табунщик Г. В. // 36. тез доп. III всеукр. наук.-техн. конф. студентів, аспірантів та молодих вчених «Інформаційні

управляючі системи та комп'ютерний моніторинг – 2012» (Донецьк, 16–18 квіт. 2012 р.). – Донецьк: ДонНТУ, 2012. – С. 363-365.

61. Брагина, Т. И. Средства обеспечения качества для моделей итеративной разработки программного обеспечения / Т.И. Брагина, Г.В. Табунщик // Тиждень науки-2010: зб. тез доповідей щорічної наук.-практ. конф. викладачів, науковців, молодих учених, аспірантів, студентів ЗНТУ, 12–16 квітня, 2010, Запоріжжя, Україна. В 4 т. Т. 2 / відп. ред. Ю. М. Внуков. – Запоріжжя : ЗНТУ, 2010. – С. 141 – 142.

62. Брагина, Т. И. Стратегия тестирования web – проектов / Брагина Т. И., Табунщик Г. В. // Наукові праці Донецького національного технічного університету. Серія “Інформатика, кібернетика та обчислювальна техніка” (ІКОТ-2012). Випуск 15 (203) – Донецьк: ДВНЗ “ДонНТУ”. – 2012.– С. 118-125.

63. Брагина, Т.И. Анализ и разработка средств тестирования web-приложений / Т.И. Брагина, Г.В. Табунщик // Интеллектуальные системы принятия решений и проблемы вычислительного интеллекта: Материалы междунар. научн. конф., Евпатория, 27-30 мая 2012г. – Херсон: ХНТУ, 2012. – С. 35-37.

64. Брагина, Т.И. Анализ подходов к управлению рисками в программных проектах с итеративным жизненным циклом / Т.И. Брагина, Г.В. Табунщик // Радиоелектроніка. Інформатика. Управління. – 2011. – №2 – С. 120-124.

65. Брагина, Т.И. Классификация методов тестирования по видам программных проектов / Т.И. Брагина, Г.В. Табунщик // Системний аналіз. Інформатика. Керування : Тези доповідей III Всеукр. науч.–практ. конф. САІУ–2012, Березень 14–16, 2012, Запоріжжя, Україна. – Запоріжжя: Вид-во КПУ. – С. 40-42.

66. Брагина, Т.И. Нечеткая модель для анализа риска программного проекта / Т.И. Брагина, Г.В. Табунщик // Системний аналіз. Інформатика. Керування: тези доповідей II Всеукр. науч.–практ. конф. САІУ–2011, Березень 10–11, 2011, Запоріжжя, Україна. Запоріжжя: Вид-во КПУ. 2011. – С. 37–38.

67. Брагина, Т.И. Нечеткий анализ проектного риска / Т.И. Брагина, Г.В. Табунщик // Системи обробки інформації. Вип.3 (93). – 2011. – С. 15-21.

68. Брагина, Т.И. Сравнительный анализ итеративных моделей разработки программного обеспечения / Т.И. Брагина,

Г.В. Табунщик // Радиоелектроніка. Інформатика. Управління. – 2010. – № 2. – С. 130 – 139.

69. Брагіна, Т.І. Средства повышения качества web – проектов / Т.І. Брагіна, Г.В. Табунщик // Радиоелектроника и молодежь в XXI веке: тези доп. 16-ого Междунар. молод. форуму, ХНУРЕ–2012, Квітень 17–19, 2012. В 6 т. – Х.:ХНУРЕ. Т. 6. 2012. – С. 135.

70. Брагіна, Т. І. Методи управління качеством информационных систем / Брагіна Т. І., Табунщик Г. В. // Тиждень науки – 2012: зб. тез доп. щоріч. наук.-практ. конф. викладачів, науковців, молодих учених, аспірантів, студентів ЗНТУ (Запоріжжя, 11–15 квіт. 2012 р.). – Запоріжжя : ЗНТУ, 2012.

71. Брагіна, Т. І. Метод оценки весовых коэффициентов при расчете трудозатрат программных проектов / Брагіна Т. І., Табунщик Г. В. // Тиждень науки – 2012: зб. тез доп. щоріч. наук.-практ. конф. викладачів, науковців, молодих учених, аспірантів, студентів ЗНТУ (Запоріжжя, 11–15 квіт. 2012 р.). – Запоріжжя: ЗНТУ, 2012.

72. Брагіна, Т. І. Комбинированный метод прогнозирования рисков / Брагіна Т. І., Табунщик Г. В. // Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій: зб. тез доп. VI міжнар. наук.-практ. конф. (Запоріжжя, 19–21 вересня 2012 р.). – Запоріжжя: ЗНТУ, 2012. – С. 265-266.

73. Брагіна, Т.І. Аналіз методів інтеграційного тестування / Т.І. Брагіна // Проблеми і перспективи розвитку ІТ-індустрії: зб. тез доповідей IV міжнар. наук.-практ. конф. (Харків, 15-16 листопада 2012 р.). – Х.: ХНЕУ, 2012.– С. 234-235.

74. Брагіна, Т.І. Розробка засобів інтеграції / Т.І. Брагіна // Системи обробки інформації. Вип.8 (106). - 2012. – С. 127-130.

75. Винниченко И. В. Автоматизация процессов тестирования / И. В. Винниченко. – СПб.: Питер, 2005. – 203 с

76. Г.Н. Калянов. Консалтинг при автоматизации предприятий: Научно-практическое издание. Серия "Информатизация России на пороге XXI века". – М.: СИНТЕГ, 1997. – 320 с.

77. Ганноцкая Т.С., Кудерметов Р.К. Анализ применения модели-ориентированного подхода для разработки сложных систем. // Тези доповідей Міжнародної науково-практичної конференції "Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій". м. Запоріжжя, 22-24 вересня 2010 р. – С. 146-147.

78. ДСТУ 2850-94. Програмні засоби ЕОМ. Показники та методи оцінювання якості. – Чинний від. 01.01.96. – К.: Дежстандарт України, 1994. – 20 с.

79. ДСТУ ISO/IEC TR 9126-2:2008. Програмна інженерія. Якість продукту. Ч. 2: Зовнішні метрики (ISO/IEC TR 9126-2:2003, IDT). – К., 2011. – VI, 85 с.

80. ДСТУ ISO/IEC TR 9126-3:2012. Програмна інженерія. Якість продукту. Частина 3. Внутрішні метрики (ISO/IEC TR 9126-3:2003, IDT)

81. ДСТУ ISO/IEC TR 9126-4:2012. Програмна інженерія. Якість продукту. Частина 4. Метрики якості під час використання (ISO/IEC TR 9126-4:2004, IDT).

82. ДСТУ ISO/IEC TR 15504-1:2002 Інформаційні технології оцінювання процесів життєвого циклу програмних засобів [Текст]; К.: Держспоживстандарт України 2004. – (Національний стандарт України). Ч. 1 : Концепції та вступна настанова (ISO/IEC TR 15504-1:1998, IDT) / пер. і наук.-техн. ред. Л. Бабко, Г. Коваль; . – Чинний від 01.10.2003. – [Б. м.] : [б.в.], 2004. – IX, 12 с.: рис.

83. ДСТУ ISO/IEC TR 15504-2:2002 Інформаційні технології оцінювання процесів життєвого циклу програмних засобів [Текст]; К.: Держспоживстандарт України 2004. – (Національний стандарт України).; Ч. 2 : Еталонна модель процесів та потужності процесу (ISO/IEC TR 15504-2:1998, IDT) / пер. і наук.-техн. ред. Л. Бабко [та ін.]; . – Чинний від 01.10.2003. – [Б. м.] : [б.в.], 2004. – VI, 33 с.

84. ДСТУ ISO/IEC TR 15504-4:2002 Інформаційні технології оцінювання процесів життєвого циклу програмних засобів [Текст]; К.: Держспоживстандарт України 2004.; Ч. 4 : Настави з виконання оцінювання (ISO/IEC TR 15504-4:1998, IDT) / пер. і наук.-техн. ред. П. Андон [та ін.]; . – Чинний від 01.10.2003. – К.: [б.в.], 2004. – VI, 17 с.

85. ДСТУ ISO/IEC TR 15504-5:2002 Інформаційні технології оцінювання процесів життєвого циклу програмних засобів [Текст]; К.: Держспоживстандарт України 2004.; Ч. 5 : Модель оцінювання та настанови щодо показників (ISO/IEC TR 15504-5:1999, IDT) / пер. і наук.-техн. ред. П. Андон [та ін.]; . – Чинний від 01.10.2003. – К.: [б.в.], 2004. – VI, 114 с.

86. ДСТУ ISO/IEC TR 15504-3:2002 Інформаційні технології оцінювання процесів життєвого циклу програмних засобів [Текст]; К.: Держспоживстандарт України 2004.; Ч. 3 : Виконання оцінювання

(ISO/IEC TR 15504-3:1998, IDT) / пер. і наук.-техн. ред. П. Андон [та ін.]; . – Чинний від 01.10.2003. – К.: [б.в.], 2004. – V, 4 с.

87. ДСТУ ISO/IEC TR 15504-9:2003 Інформаційні технології оцінювання процесів життєвого циклу програмних засобів [Текст]; К.: Держспоживстандарт України 2004.; Ч. 9 : Словник термінів (ISO/IEC TR 15504-9:1998, IDT) / пер. і наук.-техн. ред. П. Андон [та ін.]; . – Чинний від 01.07.2004. – К.: [б.в.], 2004. – IV, 11 с.

88. ДСТУ ISO/IEC TR 15504-6:2003 Інформаційні технології оцінювання процесів життєвого циклу програмних засобів [Текст]; К.: Держспоживстандарт України 2004.; Ч. 6 : Настанови з визначання компетентності оцінювачів (ISO/IEC TR 15504-6:1998, IDT) / пер. і наук.-техн. ред. П. Андон [та ін.] ; . – Чинний від 01.07.2004. – К.: [б.в.], 2004. – V, 18 с.

89. ДСТУ ISO/IEC TR 15504-7:2003 Інформаційні технології оцінювання процесів життєвого циклу програмних засобів [Текст]; К.: Держспоживстандарт України 2004. ; Ч. 7 : Настанови з удоконालювання процесу (ISO/IEC TR 15504-7:1998, IDT) / пер. і наук.-техн. ред. П. Андон [та ін.] ; . – Чинний від 01.07.2004. – К.: [б.в.], 2004. – VI, 33 с.

90. ДСТУ ISO/IEC TR 15504-8:2003 ; Інформаційні технології оцінювання процесів життєвого циклу програмних засобів [Текст]; К.: Держспоживстандарт України 2004. Ч. 8 : Настанови з визначання потужності процесу постачальника (ISO/IEC TR 15504-8:1998, IDT) / пер. і наук.-техн. ред. П. Андон [та ін.] ; . – Чинний від 01.07.2004. – К.: [б.в.], 2004. – 18 с.

91. Ефименко Н.В., Гонтарь Н.А., Кудерметов Р.К. Перспективы применения модели-ориентированного подхода в системном инжиниринге // Сборник докладов научной конференции "Информационные технологии в управлении сложными системами", 24 июня 2011 г., Днепропетровск. Изд-во "Свидлер А.Л." – С 96-99.

92. Калбертсон, Р. Быстрое тестирование / Р. Калбертсон, К. Браун, Г. Кобб. – Вильямс, 2002. – 384 с

93. Кудерметов Р.К. Дослідження ефективності роботи алгоритму планування ресурсів для розподіленої системи / Р.К. Кудерметов, Т.С. Дьячук, Р.І. Сметанін, А.М. Літава // Вісник КДУ імені Михайла Остроградського. Вип. 10 (60), частина 1, 2010. – С. 47-50.

94. Кудерметов Р.К. О критериях анализа процессов композиции веб-сервисов. / В.В. Шкарупило, Р.К. Кудерметов // Збірка матеріалів II Всеукраїнської науково-технічної конференції студентів, аспірантів та молодих вчених "Інформаційні управляючі системи та комп'ютерний моніторинг", 11-13 квітня 2011 р., Донецьк, Т.3. – С. 243-246.

95. Кудерметов Р.К. Оптимизация в системах с сервис-ориентированной архитектурой на основе свойства монотонности / Р.К. Кудерметов, М.Ю. Юрич // Наукові праці ДонНТУ. Серія "Проблеми моделювання та автоматизації проектування динамічних систем", №8 (168), 2010. – С. 161-170.

96. Кудерметов Р.К. Применение граф-аналитического подхода для распределения ресурсов виртуальной организации. / М.Б. Ильяшенко, Р.К. Кудерметов // Науковий вісник Чернівецького університету: Збірник наук. праць. Вип. 479: Комп'ютерні системи та компоненти. – Чернівці: ЧНУ, 2009. – С. 139-143.

97. Кудерметов Р.К. Природа інформації та її визначення / Р.К. Кудерметов, А.В. Притула, В.Я. Решетник. // Тези V Міжнародної науково-технічної конференції "Інформаційно-комп'ютерні технології 2010", м. Житомир, 20-22 травня 2010 р. – С. 70-72.

98. Кудерметов Р.К. Сервис-ориентированная поддержка системного инжиниринга // Електромашинобудування та електрообладнання. Міжвідомчий науково-технічний збірник. – Вип. 74, 2009. – С. 116-121.

99. Кудерметов Р.К. Сервис-ориентированная система сбора и обработки телеметрической информации / Р.К. Кудерметов, Н.Н. Хохлов, Н.Д. Маслова, О.В. Польская // Наукові праці ДонНТУ. Серія "Інформатика, кібернетика та обчислювальна техніка", №12 (165), 2010. – С. 121-125.

100. Кудерметов Р.К. Концептуальная и формальная модели систем с сервис-ориентированной архитектурой. /Р.К. Кудерметов // Науковий вісник Чернівецького університету: Збірник наук. праць. Вип. 446: Комп'ютерні системи та компоненти. – Чернівці: ЧНУ, 2009. – С.94-99.

101. Липаев, В.В. Профили стандартов жизненного цикла программных средств: метод. рекомендации. – М.: Янус-К, 2006. – 180 с.

102. Липаев, В.В. Процессы и стандарты жизненного цикла сложных программных средств: справочник. – М.: СИНТЕГ, 2006. – 276 с.

103. Липаев, В. В. Обеспечение качества программных средств. Методы и стандарты. – Синтег, Москва, 2001. – 380 с.

104. Липаев, В.В. Модели зрелости программной инженерии – СММІ. Содержание и применение / В.В. Липаев. – Информационный бюллетень, 2006. – №6(157). – С. 1-15.

105. Липаев, В.В. Методы обеспечения качества крупномасштабных программных средств. – М.: РФФИ. СИНТЕГ. 2003. – 520 с.

106. Мильман, К., Мильман С. СММІ – шаг в будущее : [Электрон. ресурс]. – Режим доступа: <http://www.osp.ru/os/2005/05-06/185610/>

107. Наши практические исследования в области расчета метрик: [Электрон. ресурс]. – Режим доступа: <http://www.viva64.com/blog/ru/2009/06/09/166/>

108. Новичков, А. Метрики кода и их практическая реализация в IBM Rational ClearCase: [Электрон. ресурс]. – Режим доступа: <http://www.viva64.com/go.php?url=241>

109. Оценка и аттестация зрелости процессов создания и сопровождения программных средств и информационных систем (ISO/IEC TR 15504 – СММ). – М.: Книга и бизнес, 2001. – 348 с.

110. Оценка и аттестация зрелости процессов создания и сопровождения программных средств и информационных систем ISO IEC TR 15504_СММІ. Пер. с англ. – М.: Книга и бизнес. 2001.

111. Паромова Т.А. Анализ влияния вида фрагментации распределенной базы данных на время выполнения запросов / Т.А. Паромова, Р.К. Кудерметов, Н.В. Луценко. // Сборник научных трудов ДонНТУ, серия: "Вычислительная техника и автоматизация". – №19 (171), 2010. – С. 168-172.

112. Розроблення методів та інформаційних технологій для моделювання процесів керування і прийняття рішень та моделей і методів оцінювання якості, надійності, відмовостійкості, живучості інформаційних систем : звіт про НДР ДР № 0111U006685 (заклучний) / Табунщик Г.В., Притула А.В., Миронова Н.О.,

Брагіна Т.І., Шитікова О.В.; кер. Г.В. Табунщик ; Запорізький національний технічний університет. – Запоріжжя, 2012. – 162 с.

113. Романов, В.Ю. Статический анализ программ: [Электрон. ресурс]. – Режим доступа: <http://www.viva64.com/go.php?url=242>

114. Скрупский, С.Ю. Имитационные модели распределенных систем компрессии видеоинформации / С.Ю. Скрупский, Р.К. Кудерметов // Наукові праці Донецького національного технічного університету. Серія: "Інформатика, кібернетика та обчислювальна техніка", №15(203), 2012. – Донецьк: ДонНТУ. – С. 190-203.

115. Табунщик Г.В. Проектування, моделювання та аналіз інформаційних систем / Кудерметов Р.К., Притула А.В., Табунщик Г.В., Запоріжжя: ЗНТУ -296 с.

116. Табунщик, Г.В. Средства поддержки agile-технологий [Текст] / Табунщик Г.В., Малюк С.С. // Зб. наук. пр V Міжнар. наук.-практ. конф. «Математичне та програмне забезпечення інтелектуальних систем» – Дніпропетровськ: ДНУ, - 2007, –С. 67

117. Табунщик, Г.В. Особливості застосування Scrum при керуванні проектами/ Г.В. Табунщик, С.С. Малюк // Тези доповідей наук. практ. конф. САІТ-2008, -С. 410.

118. Хорстман, К. Java 2. Библиотека профессионала, том 1. Основы, 7-е издание / К. Хорстман, Г. Корнелл. – М: Издательский дом "Вильямс", 2006. – 896 с.

119. Хорстман, К. Java 2. Библиотека профессионала, том 2. Тонкости программирования, 7-е издание / К. Хорстман, Г. Корнелл. – М: Издательский дом "Вильямс", 2006. – 1168с.

120. Брагіна, Т. І. модель верифікації web-орієнтованих систем [Текст] / Т. І. Брагіна, Г. В. Табунщик // ІНТЕРНЕТ-ОСВІТА-НАУКА-2014: тези доп. 19-ої міжнар. наук. – практ. конф., ІОН-2014, жовтень 14–17, 2014. – Вінниця: ВНТУ, 2014. – С. 23–25.

121. Шкарупило В.В. DEVS-модель как средство валидации композитных веб-сервисов распределенной системы / В.В. Шкарупило, С.Ю. Скрупский, Р.К. Кудерметов // Комп'ютерно-інтегровані технології: освіта, наука, виробництво. – Вип. 7, 2011. – С. 61-67.

122. Табунщик Г. В. Нові технології в підготовці фахівців з вбудованих систем / Г. В. Табунщик, А. В. Пархоменко // ІНТЕРНЕТ-

ОСВІТА-НАУКА-2014: тези доп. 19-ої міжнар. наук. – практ. конф., ІОН-2014, жовтень 14–17, 2014. – Вінниця: ВНТУ, 2014. – С.248–250.

123. Bragina T. Testing methods library for applications with web-based interfaces / T. Bragina, G. Tabunshchuk, D. Moroka // Central European Researchers Journal. – 2015. №2. – Р. 90-94.

124. Віддалений та віртуальний інструментарій в інжинірингу. Монографія. /А. Пархоменко, Г. Табунщик, М. Поляков, О. Гладкова, Т. Каплієнко, Т. Ларіонова. – Запоріжжя: Дике поле, 2015. –237 с.

АЛФАВІТНО-ПРЕДМЕТНИЙ ПОКАЖЧИК

I

ISO 37, 40

IEEE 37

M

Microsoft Solutions Framework
14

S

Scrum 23

- команда 25

- майстр 25

- практики 28

- проект 23

Software Quality Assurance - 37

Software Quality Management -
37

A

Архітектор 15

Атрибут 180

- рівня зрілості 43

- якості 77

B

Базис конфігурації 134

Бібліотекар 16

Брак 37

B

Власник продукту 25

Властивість 25

Д

Дефект 25, 125

Дефективність 125

E

Експерт з предметної області
15

Етап

- аналізу 8

- деталізоване проектування

9

- проектування 8, 14

- реалізації 14

- специфікації вимог 14

Ж

Життєвий цикл 8

журнал

- продукту 26

- спринту 26

З

Замовник 15

I

Інститут програмної інженерії
(SEI) 43

К

Керівник команди 26

Керування

- конфігурацією 135
- якістю 38

Ключові практики 45

Критерії

- якості 75

М

Менеджер проекту 15

Метрика 83

- Берлінгера 109
- Вудворда 104
- гібридна 114
- Джибла 98
- зв'язності модулів 108
- Зольновського 115
- кількісна 95
- Кокола 114
- М-міра 108
- Мартіна 110
- Мейджела 102
- Кафура 106
- міра Колофелло 109
- надійності
- об'єктно-орієнтована 114
- Пивоварського
- складності потоку

керування даними 107

- складності потоку

керування програми

- спена 106
- Чена - 102
- Холстеда 97
- цикломатичної складності

99-101

- Чидамбера і Кемерера 113
- Чепина – 104

Міра 83

- лічильні - 86
- зусиль 85
- розміру 84
- часу 85

Модель

- вартості *COCOMO* 179
- - *COCOMO II* 182
- якості 75
- ISO 9126 78
- FURPS 77
- Боема 77
- МакКола 95
- програмних проєктів 14
- Agile 19
- Crystal 22
- eXtreme Programming 20
- Feature Driven

Development 23

- Канбан 31
- Microsoft Solutions

Framework 17

- Personal Software Process /
- Team Software Process 18
- Rational Unified Process 14
- Scrum 23

П

Планувальник ресурсів 15

Проектувальник підсистеми 15

Р

Рівень

- визначений 44
- здібності

- зрілості 43
- керований 44
- оптимізований 44
- початковий 43
- повторюваний 44

Розробник 16

Розробник інформаційної
підтримки 16

С

Сертифікація 53

процес - 53

- випробувань 63

Система

- сертифікації 53

- якості 39

Стандарт

- CMMI 47
- ISO9001 40
- SPICE (ISO/IEC 15504)
- SW/CMM43

Ф

Функція втрат 69-70

\

Я

Якість 37

рівень - 37

ціна - 66

ДОДАТОК А

ІСТОРІЯ РОЗВИТКУ СИСТЕМ КЕРУВАННЯ ЯКІСТЮ

1 етап. Відповідає початковим завданням системного підходу до керування, коли з'явилася перша система – система Тейлора у 1905 році. Організаційно вона передбачала встановлення технічних і виробничих норм фахівцями та інженерами, а робочі лише зобов'язані їх виконувати. Ця система встановлювала вимоги до якості виробів у вигляді полів допусків і виводила певні шаблони, налаштовані на верхню і нижню межу допуску. Система мотивації передбачала штраф за дефекти і брак. З'явилася нова професія інспектор або технічний контролер. Взаємини з постачальниками і споживачами будувалися на основі вимог встановлених в технічних умовах, виконання яких перевірялося при приймальному контролі. Система Тейлора є системою керування якістю кожного окремо взятого виробу.

2 етап. Етап розвитку був закладений в 1924 році коли в Bell-Telephone's Laboratories розробили основи статистичного керування якістю, це були розроблення контрольних карт, перше поняття, і таблиці вибіркового контролю якості, які згодом завдяки Едварду Демінгу отримали дуже широке поширення в Японії, спираючись на точку зору Демінга Джуран, що ввів термін якість як відповідність вимогам споживача. У роботах Демінга основна увага приділяється поліпшенню якості стосовно насамперед до процесів, систем і статистики. Системи якості ускладнилися, тому що до них були додані служби, що використовують статистичні методи. З'явилася спеціальність інженер з якості, який повинен аналізувати якість і причини дефектів виробів.

3 етап. У 50-ті роки Фейгенбаумом була сформульована ідея загального контролю якості (*TQM- Total Quality Control*). До головних завдань відносять прогнозоване усунення потенційних невідповідностей продукції на стадії конструкторського розроблення. Перевірка якості продукції, комплектуючих та матеріалів, а також керування виробництвом і розвиток служби сервісного обслуговування, також була розроблена система обліку витрат на

якість. У Японії ідея *TQM* отримала розвиток в роботах Ісікави, який розглядав якість як завдання менеджменту. У той же час у Німеччині Філіп Кросбі розробив програму «0 дефектів» саме на цьому етапі з'явилися документальні системи якості, що встановлюють відповідальність і повноваження, а також взаємодію в області якості всього керівництва підприємства, а не тільки фахівців служб якості. Системи мотивації стали зміщуватися в бік людського фактора. У країнах Європи стали приділяти увагу документуванню систем забезпечення якістю та їх реєстрація або сертифікація третьою незалежною стороною.

4 етап. У 80-і роки почався перехід від загального контролю якості до загального менеджменту якості (*Total Quality Management–TQM*). Саме в цей час з'явилися міжнародні стандарти серії *ISO9000*. Якщо *TQC* – це керування якістю з метою виконання встановлених вимог, то *TQM* – це ще і керування цілями і самими вимогами. Основна ідея *TQM* базується на принципі поліпшення немає межі, основні складові *TQM* (рис. А.1).



Рис. А.1. Основні складові *TQM*

5 етап. Характеризується всесильним впливом суспільства на підприємства, це привід до появи стандартів *ISO14000*, що встановлюють вимоги до систем менеджменту з точки зору захисту навколишнього середовища та безпеки продукції.

ДОДАТОК Б

МОДЕЛІ ОЦІНКИ ВАРТОСТІ

Оцінка трудомісткості має бути ймовірнісним твердженням. Це означає, що для неї існує деякий розподіл вірогідності, який може бути дуже широким, якщо невизначеність висока, або досить вузьким, якщо невизначеність низька.

Використання власного досвіду або досвіду колег, отриманого в схожих проєктах, це найбільш прагматичний підхід, який дозволяє отримати досить реалістичні оцінки трудомісткості і терміну реалізації програмного проєкту, швидко і без великих витрат. Нереалістичність оцінок – один з серйозних демотивуючих чинників для учасників проєктної команди. Недооцінка призводить до помилок планування і неефективної взаємодії. Агресивні терміни, постійний тиск, наднормові, аврари служать причиною того, що витрати на проєкт ростуть експоненціально і необмежено.

Якщо власний досвід аналогічних проєктів відсутній, а колеги-експерти недоступні, то необхідно використати формальні методики, ґрунтовані на узагальненому галузевому досвіді. Серед них найбільше поширення отримали два підходи:

- *Functional Point Analyse International Function Point User Group (FPA IFPUG)* – метод функціональних точок,

- метод *COCOMO II (Constructive Cost Model)*.

Розглянемо, як здійснюється оцінка трудомісткості розроблення інформаційних систем в цих основних моделях. Їх використовують усі провідні фірми з розроблення ПЗ у світі для попередніх оцінок, як трудовитрат, так і бюджету проєкту. А в середині 80-х років була сформована Міжнародна Асоціація Користувачів Функціональних Точок (*IFPUG*).

Існують також інші методи для вирішення цих задач:

1. Метод функціональних точок *Mark II FPA*, що прийнятий в якості національного стандарту Великобританії, автором якого став Чарльз Саймон (*Charles Symon*), був аналогічним, але логічнішим і використав сучасну термінологію. Ця модель на відміну від *FPA IFPUG*, використовує єдине поняття транзакції, що має вхід, оброблення та вихід.

2. Метод *Feature Points*, розроблений Кэйперсом Джонсом (Capers Jones).
3. Підхід *3D Points*, розроблений в компанії Боїнг (Boeing).

Б.1 Конструктивна модель вартості

У кінці 70-х років Баррі Боемом (*Barry Boehm*) була розроблена модель оцінювання об'ємів робіт при розробці інформаційних систем, і отримала назву конструктивна модель вартості (*COCOMO*). На сьогодні ця модель оцінювання трудомісткості розроблення програмного забезпечення є найбільш відомою серед безлічі подібних моделей. Вона ґрунтується на аналізі ряду проектів, реалізованих за замовленням Міністерства оборони США. У загальних рисах вона з однієї сторони визначає відповідність між розміром системи в тисячах умовних рядків коду і " класом" проекту, а з іншої – трудомісткість розроблення системи.

Модель *COCOMO* умовно розділяє проекти тільки на три класи: обмежені, напів-інтегровані, "вбудовані системи".

Обмежені проекти – відносно невеликі розроблення, що виконуються командами, знайомими з прикладною областю. Ці проекти відрізняються відносно менш жорсткими вимогами до ПЗ, що розробляється.

Напів-інтегровані проекти характеризуються середнім розміром і складністю і виконуються групами розробників з різним досвідом. При цьому вимоги до ПЗ є жорсткішими.

Проекти «вбудованих систем» виконуються при значних апаратних, програмних і організаційних обмеженнях; є жорсткі обмеження по специфікації.

Основа COCOMO – модель, що обчислює вартість розроблення програмного забезпечення залежно від оцінок розміру коду програми і комплексу «витрат», які містять суб'єктивну оцінку товару, устаткування, персоналу та проектних характеристик. Є різні варіанти моделі *COCOMO*, які включають усі характеристики, з оцінкою вартості дій, що управляють, на кожен крок (аналіз, проектування і т.д.) в процесі розроблення програмного забезпечення.

Модель вводить 15 поправочних факторів, що належать до однієї з чотирьох категорій, які у свою чергу отримують оцінку за 6-бальною шкалою:

- **атрибути продукту**, такі, як його складність і вимоги до його надійності, розмір бази даних, складність архітектури додатка;
- **атрибути системи**, такі, як обмеження на оперативну пам'ять і час виконання, час компіляції (складання додатка), надійність використовуваних віртуальних машин;
- **атрибути команди розробників**, такі, як знання прикладної області, аналітичні здібності, досвід розроблення, досвід в даній мові програмування;
- **атрибути проекту**, такі, як використовувані засоби розроблення, застосування методів розроблення програмного забезпечення, системи контролю розроблення додатка.

З часом і зростанням вимог до систем, модель *COCOMO* виявилася застарілою в значній своїй частині. Безпосередньо з цієї причини і черги інших важливих проблем, була розроблена модель *COCOMO II*, уперше опублікована в 1999 році. Калібрування параметрів робилося згідно з 161 проектом розроблення. У моделі використовується формула регресії з параметрами, що визначаються на основі галузевих даних і характеристик конкретного проекту.

Вона удосконалює оригінальну модель в наступних основних напрямках:

- використовує вхідні дані, що доступні на ранніх етапах життєвого циклу системи для оцінки її складності (зокрема, використання функціональних точок);
- спирається на підходи, ґрунтовані на повторному використанні, що містять інтеграцію комерційних продуктів, реінжиніринг, генерацію додатків;
- використовує об'єктно-орієнтовані підходи, які підтримуються розподіленим ПЗ проміжного шару;
- враховує вплив зрілості процесів розроблення.
- містить нові – циклічні та узагальнені – моделі процесів розроблення.

Головною особливістю методики є те, що для того, щоб оцінити трудомісткість, необхідно знати розмір програмного продукту в тисячах рядках початкового коду (KSLOC). Розмір програмного

продукту може бути, наприклад, оцінений експертами із застосуванням методу PERT.

COCOMO має три рівня деталізації: базовий, проміжний і докладний. Також передбачено три режими використання моделі в залежності від розмірів команди та проекту (табл. Б.1).

Таблиця Б.1 – Режими моделі *COCOMO*

Назва режиму	Розмір проекту	Опис
Базовий	До 50 <i>KLOC</i>	Дрібний проект розробляється невеликою командою, для якої нехарактерні нововведення, і середовище залишається стабільним
Проміжний	50-300 <i>KLOC</i>	Відносно невелика команда займається проектом середнього розміру, в процесі розробки необхідні певні інновації, середовище характеризується незначною нестабільністю
Докладний	Більше 300 <i>KLOC</i>	Велика команда розробників трудиться над великим проектом, необхідний значний обсяг інновацій, середовище складається з безлічі елементів, які не характеризуються стабільністю

Для оцінки трудовитрат на базовому рівні моделі *COCOMO* застосовується наступна формула:

$$T = a \times P^b, \quad (\text{Б.1})$$

де a та b – константи, які залежать від режиму використання моделі (табл. Б.2);

P – розмір проекту в *KLOC*.

Відповідно до цієї формули трудовитрати взагалі нелінійно залежать від розміру проекту і стрибкоподібно змінюються при зміні режиму (табл. Б.2). Інша цікава особливість *COCOMO* – зростання трудовитрат при переході до більш високого режиму не означає

безумовного збільшення тривалості (F) виконання проекту, що обчислюється за формулою:

$$F = 2,5 \times T k, \quad (\text{Б.2})$$

оскільки при цьому змінюється значення константи k .

Таблиця Б.2. Значення коефіцієнтів моделі *COCOMO* в залежності від режиму використання

Назва режиму	Значення коефіцієнту a	Значення коефіцієнту b	Значення коефіцієнту k
Базовий	2,4	1,05	0,38
Проміжний	3,0	1,12	0,35
Докладний	3,6	1,20	0,32

На більш високих рівнях *COCOMO* розглянуті формули ускладнюються, використовуючи додаткові коефіцієнти, що дозволяють підвищити точність оцінок. Також модель допускає калібрування на основі хронологічних даних за виконаними проектами.

У моделі *COCOMO II* розрізняються дві стадії оцінювання проекту: попередня оцінка на початковій фазі та детальна оцінка після опрацювання архітектури.

Формула оцінки трудомісткості (PM) проекту в люд.*міс. має вигляд:

$$PM = A \times SIZE^E \times \prod_{i=1}^n EM_i, \quad (\text{Б.3})$$

где $A=2.94$, $B=0.91$;

$$E = B + 0.01 \times \sum_{j=1}^5 SF_j;$$

EM_i – множники трудомісткості;

$SIZE$ – розмір проекту в KSLOC;

SF_j – фактори масштабу;

$j=1..5$ – індекс фактору масштабу;
 $i=1..n$ – індекс множника трудомісткості;
 n – кількість множників трудомісткості;
 $n=7$ – для попередньої оцінки;
 $n=17$ – для детальної оцінки.

Фактори масштабу.

У методиці використовуються п'ять факторів масштабу SF , які визначаються наступними характеристиками проекту:

1. *PREC* – прецедентність, наявність досвіду аналогічних розробок (*Very Low* – досвід в продукті і платформі відсутній; *Extra High* – продукт і платформа повністю знайомі).

2. *FLEX* – гнучкість процесу розроблення (*Very Low* – процес строго детермінований; *Extra High* – визначені тільки спільні цілі).

3. *RESL* – архітектура і дозвіл ризиків (*Very Low* – ризики невідомі/не проаналізовані; *Extra High* – ризики дозволені на 100%).

4. *TEAM* – спрацьованість команди (*Very Low* – формальні взаємодії; *Extra High* – повна довіра, взаємозамінюваність і взаємодопомога).

5. *PMAT* – зрілість процесів (*Very Low* – CMM Level 1; *Extra High* – CMM Level 5).

Значення факторів масштабу, залежно від оцінки їх рівня, приведені у таблиці Б.3.

Таблиця Б.3. Значення факторів масштабу, залежно від оцінки його рівня

Фактори масштабу	Оцінка рівня чинника					
	Very Low	Low	Nominal	High	Very High	Extra High
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

Множники трудомісткості.

Розглянемо випадок для попереднього оцінювання трудомісткості програмного проекту. Необхідно оцінити для проекту рівень за 7 множниками трудомісткості *M*:

1. *PERS* – кваліфікація персоналу (*Extra Low* – аналітики і програмісти мають нижчу кваліфікацію, плинність більше 45%; *Extra High* – аналітики і програмісти мають вищу кваліфікацію, плинність менше 4%).

2. *RCPX* – складність і надійність продукту (*Extra Low* – продукт простий, спеціальних вимог по надійності немає, БД маленька, документацію не потрібно; *Extra High* – продукт дуже складний, вимоги по надійності жорсткі, БД надвелика, документацію потрібно в повному об'ємі).

3. *RUSE* – розробка для повторного використання (*Low* – не вимагається; *Extra High* – потрібно перевикористання в інших продуктах).

4. *PDIF* – складність платформи розроблення (*Extra Low* – спеціальні обмеження по пам'яті і швидкодії відсутні, платформа стабільна; *Extra High* – жорсткі обмеження по пам'яті і швидкодії, платформа нестабільна).

5. *PREX* – досвід персоналу (*Extra Low* – нове застосування, інструменти і платформа; *Extra High* – додаток, інструменти і платформа добре відомі).

6. *FCIL* – устаткування (*Extra Low* – інструменти прості, комунікації ускладнені; *Extra High* – інтегровані засоби підтримки життєвого циклу, інтерактивні мультимедіа комунікації).

7. *SCED* – стискування розкладу (*Very Low* – 75% від номінальної тривалості; *Very High* – 160% від номінальної тривалості).

Вплив множників трудомісткості залежно від їх рівня визначається їх числовими значеннями, які представлені в матриці, приведеній нижче (табл. Б.4).

З цієї таблиці, зокрема, слідує, якщо в нашому проекті низька кваліфікація аналітиків, то його трудомісткість зросте приблизно в 4 рази в порівнянні з проектом, в якому беруть участь аналітики екстра-класу.

Таблиця Б.4. Значення множників трудомісткості, залежно від оцінки їх рівня

	Оцінка рівня множника трудомісткості						
	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
<i>PERS</i>	2.12	1.62	1.26	1.00	0.83	0.63	0.5
<i>RCPX</i>	0.49	0.60	0.83	1.00	1.33	1.91	2.72
<i>RUSE</i>	n/a	n/a	0.95	1.00	1.07	1.15	1.24
<i>PDIF</i>	n/a	n/a	0.87	1.00	1.29	1.81	2.61
<i>PREX</i>	1.59	1.33	1.22	1.00	0.87	0.74	0.62
<i>FCIL</i>	1.43	1.30	1.10	1.00	0.87	0.73	0.62
<i>SCED</i>	n/a	1.43	1.14	1.00	1.00	1.00	n/a

На приклад для проекту були обрані фактори масштабу:

- PREC «Very High» (оцінка 1,24);
- FLEX «Very Low» (оцінка 5,07);
- RESL «High» (оцінка 2,83);
- TEAM «High» (оцінка 2,19);
- PMAT «Low» (оцінка 6,24);
- *SIZE* = 10KLOC ;
- EAF=1.

Тоді розрахуємо згідно з (Б.4):

$$E = B + 0.01 \times \sum_{j=1}^5 SF_j = 1.0857 .$$

Розраховуємо попередню оцінку трудомісткості проекту:

$$PM = 2,94 \times 10^{1,0857} = 35,8 \text{ людино-місяців.}$$

З використанням множників трудомісткості (табл. Б.5) трудомісткість розрахується наступним чином:

$$PM = 35,8 \times 1,91 \times 1,81 \times 0,5 = 61,9 \text{ людино-місяців.}$$

Таблиця Б.5. Приклад значень множників трудомісткості

Множник трудомісткості	Оцінка рівня множника трудомісткості	
<i>PERS</i>	Extra High	0.5
<i>RCPX</i>	Very High	1.91
<i>RUSE</i>	Nominal	1.00
<i>PDIF</i>	Very High	1.81
<i>PREX</i>	Nominal	1.00
<i>FCIL</i>	Nominal	1.00
<i>SCED</i>	Nominal	1.00

Таким чином, формула з використанням множників трудомісткості надає результат, який майже у 2 рази відрізняється від попередньої оцінки.

Оцінка багатокомпонентного продукту

Для того, щоб адекватно спланувати проект і оцінити його трудомісткість, необхідно виконати попереднє проектування програмного продукту. В результаті декомпозиції ми отримуємо деяку кількість компонентів (N), які складають програмний продукт.

Слід розуміти, що сумарна трудомісткість проекту не дорівнює простій сумі трудомісткостей розроблення кожного з компонентів:

$$PM \neq \sum_{k=1}^N PM_k ,$$

де PM_k – трудомісткість k -ого компоненту, $k=1,..N$.

Проста сума не враховує взаємозв'язки компонентів і трудовитрати на їх інтеграцію.

Методика *COCOMO II* визначає наступну послідовність обчислення трудомісткості проекту при багатокомпонентній розробці.

1. Сумарний розмір продукту розраховується, як сума розмірів його компонентів:

$$SIZE^A = \sum_{k=1}^N SIZE_k ,$$

де $SIZE_k$ _ розмір k -ого компоненту;

N – кількість компонент проекту.

2. Базова трудомісткість проекту розраховується по формулі:

$$PM^B = A \times (SIZE^A)^E \times SCED,$$

де A – розмір k -ого компоненту;

$A=2.94$, $B=0.91$;

$$E = B + 0.01 \times \sum_{j=1}^5 SF_j ;$$

$SCED$ – можливість стискування розкладу.

3. Потім розраховується базова трудомісткість кожного компонента:

$$PM_k^B = PM^B \times \frac{SIZE_k}{SIZE^A}.$$

4. На наступному кроці розраховується оцінка трудомісткості компонентів з урахуванням усіх множників трудомісткості, окрім множника $SCED$:

$$PM_k' = PM_k^B \times \prod_{i=1}^6 EM_i, \quad (Б.5)$$

де EM_i – множники трудомісткості (PERS, RCPX, RUSE, PDIF, PREX, FCIL, SCED).

5. І, нарешті, підсумкова трудомісткість проекту визначається за формулою:

$$PM = \sum_{k=1}^N PM_k'. \quad (Б.6)$$

Оцінка тривалості проекту

Тривалість проекту в методиці *COCOMO II* розраховується за формулою:

$$T_{dev} = C \times (PM_{NS})^{D+0.2 \times 0.01 \times \sum_{j=1}^5 SF_j} \times \frac{SCED}{100}, \quad (Б.7)$$

де $C = 3,67$; $D = 0,28$;

PM_{NS} – трудомісткість проекту без урахування множника SCED, що визначає стискування розкладу.

Для програмних проектів, зайнятих розробкою комерційних додатків, метод *COCOMO* зазвичай призводить до завищених значень оцінки витрат. Тому метод *COCOMO* застосовується тільки до розробки технічного програмного забезпечення.

Це обставина пов'язана з тим, що відношення DSI до людино-місяців, реалізоване в оціночному рівнянні методу *COCOMO*, відповідає продуктивності при розробці технічного програмного забезпечення. У відношенні комерційної розробки програмного забезпечення покладаються більш високі значення показника *DSI* / людино-місяць.

Б.2 Методологія функціональних точок для оцінювання вартості розроблення програмного забезпечення

Напевно, найбільш вдалою заміною кількості рядків коду для вимірювання продуктивності стали функціональні точки (function points), вперше запропоновані співробітником IBM Аланом Альбрехтом (Allan Albrecht) в 1979 р. Найважливіша їх перевага полягає в тому, що оскільки застосування функціональних точок засноване на вивченні вимог, то оцінка необхідних трудовитрат може бути виконана на самих ранніх стадіях роботи над проектом і далі буде уточнюватися з розвитком проекту.

Поступово метод функціональних точок перетворився на індустріальний стандарт, і в 1986 р. для його підтримки та розвитку була створена некомерційна організація IFPUG. Завдання методу полягає в тому, щоб розбити систему на дрібні частини для кращого розуміння та аналізу, що забезпечує більш структурований підхід для вирішення проблеми.

Метод, призначений для оцінки трудомісткості проекту, який на основі логічної моделі оцінює обсяг програмного продукту кількістю функціоналу, що потребує замовник та поставляє розробником. Безсумнівною перевагою методу є те, що вимірювання не залежать від технологічної платформи, на якій буде розроблятися продукт, і він забезпечує однаковий підхід до оцінки всіх проектів в компанії.

При аналізі методом функціональних точок треба виконати наступну послідовність кроків:

1. Визначити тип оцінки трудомісткості проекту.
2. Визначити області оцінки та меж продукту.
3. Розрахувати функціональні точки, пов'язані з даними.
4. Розрахувати функціональні точок, пов'язані з транзакціями.
5. Визначити сумарну кількість не вирівняних функціональних точок (*UFP*).
6. Визначити значення фактору вирівнювання (*FAV*).
7. Розрахувати кількість вирівняних функціональних точок (*AFP*).

Перший крок полягає у визначенні об'єкту оцінювання. Метод передбачає оцінювання трьох об'єктів :

- «проект розробки», для якого оцінюється кількість функцій, що поставляються користувачам в першому релізі продукту.
- «проект розвитку», для якого оцінюється в функціональних точках об'єм доопрацювання: додавання, змінювання та видалення функціоналу.
- «продукт», для якого оцінюється обсяг вже існуючого та розгорнутого продукту.

Другий крок – це визначення області оцінювання та меж продукту.

В залежності від типу область оцінювання може містити:

- всі функції, що розробляються (для проекту розробки);
- всі додані, змінені і видалені функції (для проектів розвитку);
- тільки функції, які реально використовуються, або всі функції (при оцінці продукту та/або продуктів).

На другому кроці також визначають межі продукту:

- що є «зовнішнім» відносно оцінюваного продукту?

– де розташовується «межа системи», через яку проходять операції, що передаються або приймаються системою, з точки зору користувача?

– які дані підтримуються додатком, а які є зовнішніми для додатку?

До логічних даних системи відносяться:

1. Внутрішні логічні файли (*Internal logical files – ILFs*) – виділяються користувачем логічно пов'язані групи даних або блоки керуючої інформації, які підтримуються всередині продукту.

2. Зовнішні інтерфейсні файли (*External interface files – EIFs*) – виділяються користувачем у логічно пов'язані групи даних або блоки керуючої інформації, на які посилається продукт, але які підтримуються поза продуктом.

Прикладом логічних даних (інформаційних об'єктів) можуть служити: клієнт, рахунок, тарифний план, послуга.

Третій крок – підрахунок функціональних точок, пов'язаних з даними. Спочатку визначається складність даних за наступними показниками:

1. *DET (Data Element Type)* – неповторюване унікальне поле даних, наприклад, Ім'я Клієнта – 1 *DET*; Адреса Клієнта (індекс, країна, область, район, місто, вулиця, будинок, корпус, квартира) – 9 *DET*'s.

2. *RET (Record Element Type)* – логічна група даних, наприклад, адреса, паспорт, телефонний номер.

Оцінка кількості не вирівняних функціональних точок, залежить від складності даних, яка визначається на підставі матриці складності (Таблиця Б.6).

Таблиця Б.6. Матриця складності даних

Кількість <i>DET</i> \ Кількість <i>RET</i>	1-19 <i>DET</i>	20-50 <i>DET</i>	50+ <i>DET</i>
1 <i>RET</i>	Low	Low	Average
2-5 <i>RET</i>	Low	Average	High
6+ <i>RET</i>	Average	High	High

Оцінка даних в не вирівняних функціональних точках (*UFP*) підраховується по-різному для внутрішніх логічних файлів (*ILFs*) і для зовнішніх інтерфейсних файлів (*EIFs*) (Таблиця Б.7) залежно від їх складності.

Таблиця Б.7. Оцінка даних в не вирівняних функціональних точках (*UFP*) для внутрішніх логічних файлів (*ILFs*) і зовнішніх інтерфейсних файлів (*EIFs*)

Складність даних	Кількість <i>UFP (ILF)</i>	Кількість <i>UFP (EIF)</i>
<i>Low</i>	7	5
<i>Average</i>	10	7
<i>High</i>	15	10

Для ілюстрації розглянемо приклад оцінки в не вирівняних функціональних точках об'єкта даних «Клієнт» (Рис. Б.1).

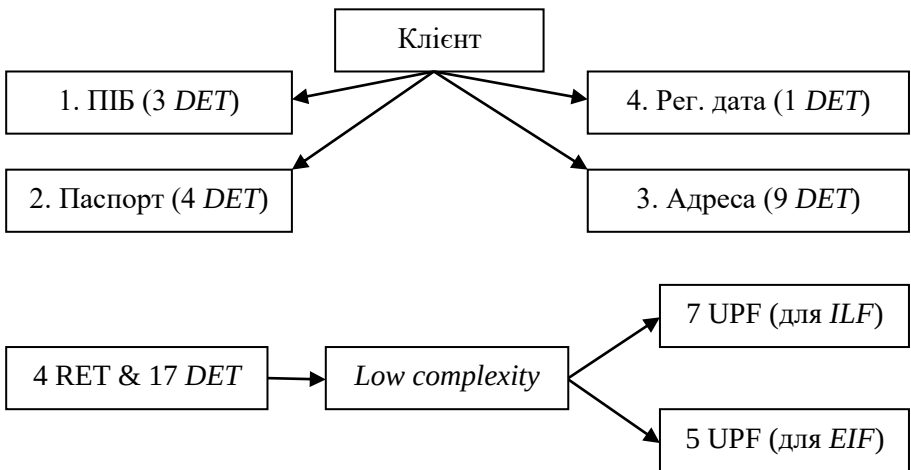


Рис.Б.1. Приклад оцінки в не вирівняних функціональних точках об'єкта даних «Клієнт»

Об'єкт «Клієнт» містить чотири логічних групи даних, які в сукупності складаються з 15 неповторюваних унікальних полів даних. Згідно матриці (Таблиця Б.6), нам варто оцінити складність цього

об'єкта даних, як «Low». Тепер, якщо оцінюваний об'єкт відноситься до внутрішніх логічних файлів, то згідно таблиці Б.7 його складність буде – 7 не вирівняних функціональних точок (*UPF*). Якщо ж об'єкт є зовнішнім інтерфейсним файлом, то його складність складе 5 *UPF*.

Підрахунок функціональних точок, пов'язаних з транзакціями – це четвертий крок аналізу за методом функціональних точок.

Транзакція – це елементарний неподільний замкнутий процес, що представляє значення для користувача і переводить продукт з одного стану в інший.

У методі розрізняють такі типи транзакцій (Таблиця Б.8):

– зовнішні входні транзакції *EI* (*external inputs*) – елементарна операція з обробки даних або керуючої інформації, що надходить до системи зовні.

– зовнішні вихідні транзакції *EO* (*external outputs*) – елементарна операція з генерації даних або керуючої інформації, що виходять за межі системи. Припускає певну логіку обробки або обчислень інформації з одного або більше *ILF*.

– зовнішні запити *EQ* (*external inquiries*) – елементарна операція, яка у відповідь на зовнішній запит витягує дані або керуючу інформацію з *ILF* або *EIF*.

У таблиці Б.8 наведені основні між типами транзакцій.

Таблиця Б.8. Основні відмінності між типами транзакцій.
Легенда: О – основна; Д – додаткова; *NA* – не застосовна.

Функція	Тип транзакції		
	<i>EI</i>	<i>EO</i>	<i>EQ</i>
Змінює поведінку системи	О	Д	<i>NA</i>
Підтримка одного або більше <i>ILF</i>	О	Д	<i>NA</i>
Представлення інформації користувачеві	Д	О	О

Оцінка складності транзакції ґрунтується на наступних її характеристиках:

1. *FTR* (*File Type Referenced*) – дозволяє підрахувати кількість різних файлів (інформаційних об'єктів) типу *ILF* і/або *EIF*, що модифікуються або зчитуються в транзакції.

2. *DET (Data Element Type)* – кількість неповторюваних унікальних полів даних. Приклади. *EI*: поле введення, кнопка. *EO*: поле даних звіту, повідомлення про помилку. *EQ*: поле введення для пошуку, поле виведення результату пошуку.

Для оцінювання складності транзакцій служать матриці в таблицях Б.9 та Б.10.

Таблиця Б.9. Матриця складності зовнішніх вхідних транзакцій (*EI*)

<i>EI</i>	<i>1-4 DET</i>	<i>5-15 DET</i>	<i>16+ DET</i>
<i>0-1 FTR</i>	<i>Low</i>	<i>Low</i>	<i>Average</i>
<i>2 FTR</i>	<i>Low</i>	<i>Average</i>	<i>High</i>
<i>3+ FTR</i>	<i>Average</i>	<i>High</i>	<i>High</i>

Таблиця Б.10. Матриця складності зовнішніх вихідних транзакцій і зовнішніх запитів (*EO & EQ*)

<i>EO & EQ</i>	<i>1-5 DET</i>	<i>6-19 DET</i>	<i>20+ DET</i>
<i>0-1 FTR</i>	<i>Low</i>	<i>Low</i>	<i>Average</i>
<i>2-3 FTR</i>	<i>Low</i>	<i>Average</i>	<i>High</i>
<i>4+ FTR</i>	<i>Average</i>	<i>High</i>	<i>High</i>

Оцінка транзакцій в *UFP* може бути отримана з матриці (таблиця Б.11)

Таблиця Б.11. Складність транзакцій в не вирівняних функціональних точках (*UFP*)

Складність транзакцій	Кількість <i>UFP (EI & EQ)</i>	Кількість <i>UFP (EO)</i>
<i>Low</i>	3	4
<i>Average</i>	4	5
<i>High</i>	6	7

Загальний обсяг продукту в *UFP* визначається шляхом підсумовування по всіх інформаційним об'єктам (*ILF*, *EIF*) і елементарним операціям (транзакцій *EI*, *EO*, *EQ*):

$$UFP = \sum_{ILF} UFP_i + \sum_{EIF} UFP_i + \sum_{EI} UFP_i + \sum_{EO} UFP_i + \sum_{EQ} UFP_i. \quad (Б.8)$$

Крім функціональних вимог на продукт накладаються загальносистемні вимоги, що обмежують розробників у виборі рішення і збільшують складність розробки. Для обліку цієї складності застосовується фактор вирівнювання (*VAF*).

Розрахунок значення фактора вирівнювання здійснюється за формулою

$$VAF = (TDI * 0.01) + 0.65.$$

Розрахунок сумарного ефекту 14 системних характеристик (*total degree of influence, TDI*) здійснюється простим підсумовуванням:

$$TDI = \sum DI, \quad (Б.9)$$

де *DI (degree of influence)* – ступінь впливу системної характеристики, та оцінюються за шкалою від 0 до 5.

Значення фактора *VAF* залежить від 14 параметрів *DI*, які визначають системні характеристики продукту:

1. Обмін даними (0 – продукт являє собою автономне додаток; 5 – продукт обмінюється даними по більш, ніж одному телекомунікаційному протоколу).

2. Розподілена обробка даних (0 – продукт не переміщує дані; 5 – розподілена обробка даних виконується декількома компонентами системи).

3. Продуктивність (0 – вимоги користувача з продуктивності не встановлені; 5 – час відгуку сильно обмежена критично для всіх бізнес-операцій, для задоволення вимогам необхідні спеціальні проектні рішення та інструменти аналізу).

4. Обмеження по апаратних ресурсів (0 – немає обмежень; 5 – продукт цілком повинен функціонувати на певному процесорі і не може бути розподілений).

5. Транзакційна навантаження (0 – транзакцій не багато, без піків; 5 – кількість транзакцій велика і нерівномірна, потрібні спеціальні рішення та інструменти).

6. Інтенсивність взаємодії з користувачем (0 – всі транзакції обробляються в пакетному режимі; 5 – більше 30% транзакцій – інтерактивні).

7. Ергономіка (ефективність роботи кінцевих користувачів) (0 – немає спеціальних вимог; 5 – вимоги щодо ефективності дуже жорсткі).

8. Інтенсивність зміни даних (*ILF*) користувачами (0 – не вимагаються; 5 – зміни інтенсивні, жорсткі вимоги по відновленню).

9. Складність обробки (0 – обробка мінімальна; 5 – вимоги безпеки, логічна і математична складність, багатопоточність).

10. Повторне використання (0 – не вимагається; 5 – продукт розробляється як стандартний багаторазовий компонент).

11. Зручність інсталяції (0 – немає вимог; 5 – установка і оновлення ПЗ проводиться автоматично).

12. Зручність адміністрування (0 – не вимагається; 5 – система автоматично самовідновлюється).

13. Портуючість (0 – продукт має тільки 1 інсталяцію на єдиному процесорі; 5 – система є розподіленою і припускає установку на різні «залізо» і ОС).

14. Гнучкість (0 – не вимагається; 5 – гнучка система запитів і побудова довільних звітів, модель даних змінюється користувачем в інтерактивному режимі).

Наприклад, якщо, кожен з 14 системних параметрів отримав оцінку 3, то їх сумарний ефект складе $TDI = 3 * 14 = 42$. У цьому випадку значення фактора вирівнювання буде: $VPF = (42 * 0.01) + 0.65 = 1.07$

Подальша оцінка в вирівняних функціональних точках залежить від типу оцінки. Початкова оцінка кількості вирівняних функціональних точок для програмного додатка визначається за наступною формулою:

$$AFP = UFP * VPF, \quad (Б.10)$$

де *UFP* – загальний обсяг продукту в не вирівняних функціональних точках (Б.4).

Вона враховує тільки нову функціональність, яка реалізується в продукті.

«Проект розробки» продукту оцінюється в *DFP* (*development functional point*) за формулою:

$$DFP = (UFP + CFP) * VAF, \quad (Б.11)$$

де *CFP* (*conversion functional point*) – функціональні точки, підраховані для додаткової функціональності, яка буде потрібна при розгортанні продукту, наприклад, міграції даних.

«Проект доопрацювання і вдосконалення продукту» оцінюється в *EFP* (*enhancement functional point*) за формулою:

$$EFP = (ADD + CHG_A + CFP) * VAF_A + (DEL * VAF_B), \quad (Б.12)$$

де *ADD* – функціональні точки для доданої функціональності;

CHG_A – функціональні точки для змінених функцій, розраховані після модифікації;

VAF_A – величина фактора вирівнювання розрахованого після завершення проекту;

DEL – обсяг вилученої функціональності;

VAF_B – величина фактора вирівнювання розрахованого до початку проекту.

Сумарний вплив процедури вирівнювання лежить в межах $\pm 35\%$ відносно обсягу розрахованого в *UFP*.

Метод аналізу функціональних точок нічого не говорить про трудомісткість розробки оціненого продукту. Питання вирішується просто, якщо компанія розробник має власну статистику трудовитрат на реалізацію функціональних точок. Якщо такої статистики немає, то для оцінки трудомісткості і термінів проекту можна використовувати метод *SOCOMO II*.

Якщо аналіз продукту був виконаний методом функціональних точок, то його розмір може бути розрахований з використанням власних статистичних даних або з використанням статистики по галузі (таблиця Б.12).

Таблиця Б.12. Оцінка кількості рядків, необхідних на реалізацію однієї не вирівняної функціональної точки для деяких поширених мов програмування.

Мова програмування	Оцінка кількості рядків		
	Оптимістична	Оптимістична	Оптимістична
<i>Assembler</i>	172	86	320
<i>C</i>	148	9	704
<i>C++</i>	60	29	178
<i>C#</i>	59	51	66
<i>J2EE</i>	61	50	100
<i>JavaScript</i>	56	44	65
<i>PL/SQL</i>	46	14	110
<i>Visual Basic</i>	50	14	276

ДОДАТОК В

ПЕРЕЛІК ТЕМ ДЛЯ ПРАКТИЧНИХ ЗАВДАНЬ

1. Розробка електронного журналу успішності.
2. Розробка аналогу GuitarPro.
3. Розробка електронного гаманця.
4. Розробка переглядача електронних книг.
5. Розробка відеоплеєра.
6. Розробка графічного редактору.
7. Розробка програмного забезпечення для віртуальної реальності.
8. Розробка програмного забезпечення для тестування в ДАІ.
9. Розробка аудіоплеєра.
10. Розробка навігатора.
11. Розробка системи сканування QR-кодів.
12. Розроблення браузеру.
13. Розробка системи моніторингу стану комп'ютеру.
14. Розробка аналогу PlayMarket.
15. Розробка HTML-редатора.
16. Розробка браузерної гри.
17. Розробка графічної оболонки для операційної системи.
18. Розробка програми для відеомонтажу.
19. Розробка системи для дитячого розвитку.
20. Розробка системи для блокування реклами.
21. Розробка системи для моделювання одягу (взуття).
22. Розробка аналогу GeoGuesser (програма для орієнтації на місцевості).

Навчальне видання

**ТАБУНЩИК Галина Володимирівна
КУДЕРМЕТОВ Равіль Камілович
КАПЛІЄНКО Тетяна Ігорівна**

**ІНЖЕНЕРІЯ ЯКОСТІ ПРОГРАМНОГО
ЗАБЕЗПЕЧЕННЯ
2-ге видання**

Навчальний посібник

Комп'ютерний набір
Дизайн обкладинки
Технічний редактор
Коректор
Художник

Г.В. Табунщик
А.О. Польська
Л. А. Рябоконь
Н. В. Чечєко
А. П. Кондаков

Формат 60x84/16.

Папір офсетний. Гарнітура *Times*. Друк офсетний.
Підписано до друку 15.04.2016. Ум. друк. арк. 10,23.

Видавництво «Дике Поле»

Україна, 69063, м. Запоріжжя, вул. Троїцька, 31-А.

Тел.: (061) 213-75-95; 213-75-05.

Свідомство суб'єкта видавничої справи 33 № 004 від 23.08.2001 р.

Табунщик Г.В., Кудерметов Р.К., Каплієнко Т.І.

Т-12 Інженерія якості програмного забезпечення: навчальний посібник, 2-ге видання. – Запоріжжя: Дике Поле, 2016.. – 176 с.

ISBN 978-966-2752-94-6

Книга містить основні концепції керування якістю, зокрема якістю програмного забезпечення, на протязі всього життєвого циклу. Розглянуті питання сертифікації та оцінки витрат на якість. Особливу увагу приділено метрикам якості інформаційних систем та їх використанню. Видання призначене для студентів комп'ютерних спеціальностей вищих навчальних закладів, а також може бути використаним аспірантами, науковими та педагогічними працівниками, фахівцями-практиками.

УДК 004.41 (075.8)

ББК 32.972–02 я73