STEP IT ACADEMY

КЕРУВАННЯ ПРОГРАМНИМИ ПРОЄКТАМИ

УРОК 1

ВСТУП ДО КЕРУВАННЯ ПРОГРАМНИМИ ПРОЄКТАМИ

ЗМІСТ

1. Вступ до предметної області 4

Причини виникнення дисципліни «Керування програмними проєктами» 4

Діаграми Ґанта 6

Що таке проєкт та програмний проєкт? 8

Визначення інжинірингу ПЗ 9

Визначення проєкту 10

Що таке керування проєктами? 12

Що таке командна розробка? 13

Аналіз проблем одиночної та командної розробки програмного забезпечення 16

Аналіз термінів предметної галузі 17

Характеристики проєкту 22

Витрати, пов'язані з проєктом 27

2. Моделі та методологія процесу розробки 29
Загальний огляд моделей та методологій процесу розробки 29
Фази процесу 29
Фази процесу розробки. 30
Водоспадна модель 35
Макетування 36
Спіральна модель 38
Компонентно-орієнтована модель 40
Ітеративна модель 40
Аналіз існуючих моделей та методів 49
3. Управління якістю. 51
Історія розвитку систем якості 52
Що таке якість? Фактори якості 55
4. Документування 57
Стандартизація документації 62

1. ВСТУП ДО ПРЕДМЕТНОЇ ОБЛАСТІ

Причини виникнення дисципліни «Керування програмними проєктами»

Аналіз розробки ПЗ показав, що велика кількість проєктів розробляється з відхиленнями від технічного завдання на проєкт, термінів реалізації проєкту і завжди виходить за рамки бюджету. Причин багато. Назвемо деякі разом із наслідками.

1. **Замовник не представляє можливостей розробки та застосування.**

   Наслідок: в процесі розробки у замовника з'являються нові погляди на те, що він хотів би отримати.

2. **Замовник не розуміє складності розробки.**

   Наслідок: бюджет і терміни виконання встановлюються абсолютно нездійсненними.

3. **Виконавець не знає предметної області і не в змозі оцінити складність завдання.**

   Наслідок: затягування термінів, непомірне роздування бюджету, порушення бюджету і термінів, неправильне рішення або взагалі його відсутність.

4. **Зміна існуючого стану речей**, починаючи від виникнення нових методів та технологій розробки, або заміни технологій у замовника, і закінчуючи крахом фірми замовника або виконавця.

Можливі різні комбінації цих та інших причин, що призводить до величезного різноманіття пояснень краху

більшості проєктів (за оцінками – до 30% провалилися і всього близько 30% завершених). Це призвело до необхідності страхування як замовника, так і виконавця від неуспішного завершення, яку надає комп'ютерна інженерія. Однією зі складових комп'ютерної інженерії є дисципліна **«Керування проєктами»**. У керуванні проєктами мова йде про організацію процесу розробки ПЗ. Процес розробки ПЗ можна розбити на кілька складових частин, які називаються **життєвим циклом проєкту** (рис. 1).

Стадії життєвого циклу:

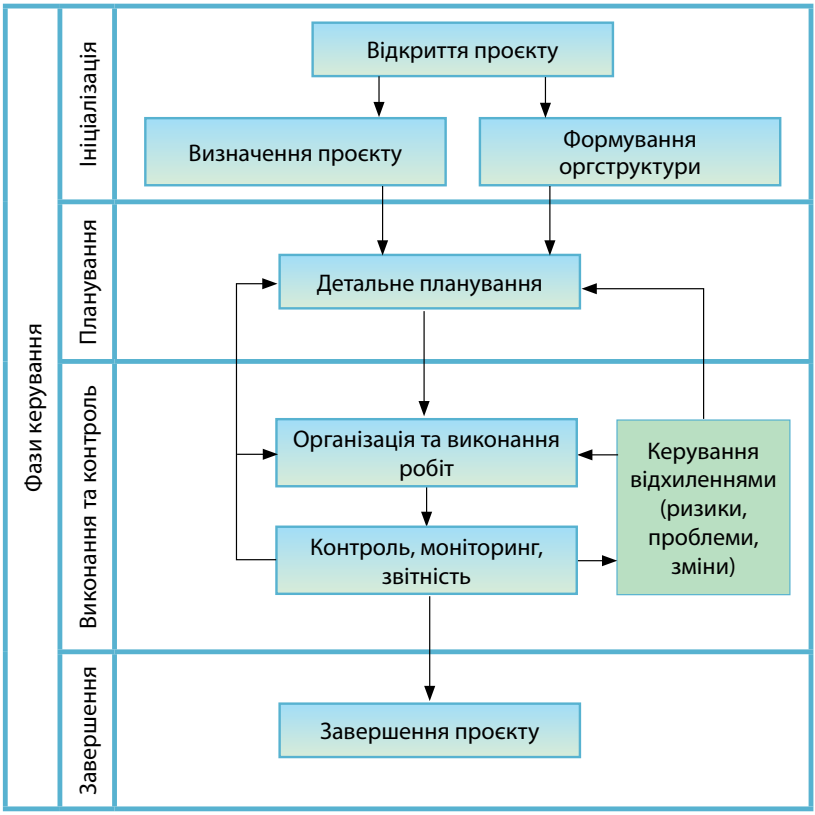


Рисунок 1

На рисунку 1 показано, що стадії життєвого циклу програми поділяються на неумовні фази, кожна з яких має призначення. Відтак, фаза відкриття проєкту часто називається фазою намірів сторін. Фаза планування раніше називалася розробкою технічного завдання, яке було непорушним до закінчення проєкту. Рисунок показує, що зараз передбачено вплив фази виконання на фазу планування, яка в такому випадку вже називається перепланування̆м.

Діаграми Ґанта

Процес розробки можна розбити на складові частини, для кожної з них визначити тривалість і послідовність виконання. При цьому розглядаються дії, послідовність яких незмінна щодо одна одної. Наприклад, не можна розробити форму для введення властивостей об'єкта, якщо ми не визначили, які властивості має цей об'єкт. Або простіший приклад: не можна налагодити частину програми, яка ще не написана.

Для відображення послідовних складових частин будь-якого процесу зручно користатися **діаграмами Ґанта** (*дослідник процесів виробництва початку XX століття, перша діаграма була запропонована у 1910 році – понад 100 років тому!*) – горизонтальні стовпчасті діаграми, кожен новий вид робіт відображається в новому рядку.

Наприклад, напишемо план виконання домашнього завдання (рис. 2):

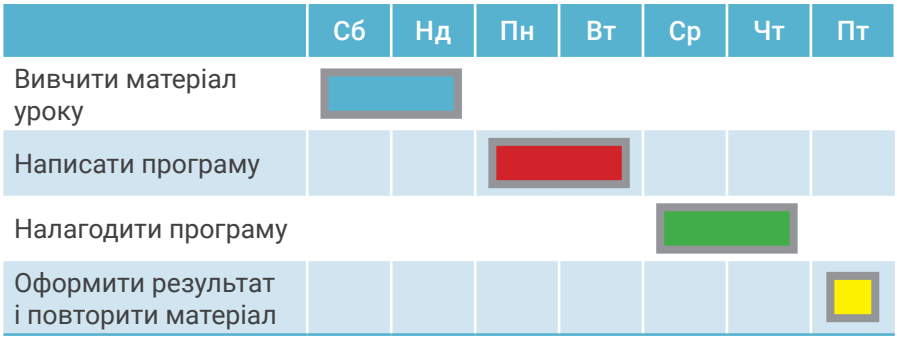


Рисунок 2

Діаграми Ганта використовуються для складання планів. Кожен стовпчик відповідає окремій дії. Дії, що становлять план, розміщуються по вертикалі. Початок, кінець і довжина стовпця на шкалі часу відповідають початку, кінцю та тривалості роботи. Можна вказувати залежність між роботами. Діаграма може використовуватися для представлення поточного стану виконання робіт: частина прямокутника, яка відповідає виконаній частині роботи, заштриховується. Вертикальною лінією відзначають поточний момент. До діаграми надається таблиця зі списком робіт, рядки якої відповідають окремо дії з діаграми, а стовпці містять додаткову інформацію про заплановану роботу.

Що таке проєкт та програмний проєкт?

Програмне забезпечення (ПЗ) – це програма або група програм , яка є кінцевим продуктом проєкту програмної розробки. Програмну розробку називають програмним інжинірингом.

Інститут програмного інжинірингу (Software Engineering Institute, SEI) дав таке визначення програмного забезпечення:

*«**ПЗ** – програми, процедури та мови символів, які керують функціонуванням апаратних засобів».*

Проєкт – це заздалегідь спланована послідовність дій.

Інститут управління проєктами (Project Management Institute, PMI) дав таке визначення проєкту:

*«**Проєкт** це:*

- *Певний план або розробка проєкту: схема*;
- *Запланована дія:*
  (а) певним чином сформульована частина дослідження;
  (б) велика, зазвичай підтримана субсидіями деяких організацій або уряду, дія;
  (в) завдання або проблема, зазвичай поставлена перед групою студентів, яка потім виноситься на аудиторні заняття».

Ще одне визначення – визначення менеджменту:

***Менеджмент** – це практика виконання проєкту та керування проєктом.*

Визначення інжинірингу ПЗ

Згідно з Баррі Боемом *(Barry Boehm):*

*«**Інжиніринг ПЗ** – це практичне застосування наукових знань при розробці та створенні комп'ютерних програм, та пов'язаної з ними документації, необхідної для їх розробки, використання та підтримки».*

(Boehm B. @Software Engineering Economics», Englewood Cliffs, NJ: Prentice Hall, 1976.– p.16).

Інститут IEEE дає таке визначення інжинірингу:

*«**Інжиніринг ПЗ** – систематичний підхід до розвитку, дії, підтримки та припинення експлуатації ПЗ».*

(Institute of Electrical and Electronics Engineers. IEEE Std 610.12.1990 Standard Glossary of Software Engineering Terminology. «Software Engineering Collection». NY: Institute of Electrical and Electronics Engineers, 1983.– p.76).

Згідно зі Стефаном Шахом (Stephen Schach):

*«**Програмний інжиніринг** – дисципліна, метою якої є створення якісного ПЗ, яке завершується вчасно, не веде до перевищення виділених бюджетних коштів, а також задовольняє висунутим вимогам».*

(Schach Stephen R. «Classical and Object-Oriented Software Engineering» 4th ed. Boston, MA: McGrawHill, 1999.– p.4).

Якщо об'єднати ці визначення, то вийде визначення, яке найбільш повно відображає суть:

***Інжиніринг програмного забезпечення** – це регламентована, системна методологія розробки, використання, обслуговування та припинення експлуатації програмного забезпечення на основі практичного застосування наукових знань.*

Визначення проєкту

Визначення, подане Джеймсом (James Lewis):

*«**Проєкт** – одноразова робота, яка має певні дати початку та закінчення, чітко визначені цілі, можливості та, якправило, бюджет. Це дії, що відрізняються від повторюваних операцій, таких як виробництво, обробка замовлення і т. д. У даному випадку йдеться про специфічну дію, що має певні тактичні цілі».*

(Lewis James P. «Project Planning, Schedulling and Control: A Hands-On Guide to Bringing Projects in on Time and on Budget», rev ed. Chicago, IL: Irwin, 1995.– pp. 2–3).

*За Гарольдом Керцнером (Harold Kerzner) **проєкт це** –*

«Довільний ряд дій або завдань, що має певну мету, яка буде досягнута в рамках виконання деяких завдань, що характеризуються певними датами початку та закінчення, межами фінансування (у разі прикладного проєкту) та ресурсами (гроші, трудові витрати, обладнання)».

(Kerzner H. «Project Management: A System Approach to Planning, Schedulling and Controlling» 6th ed. NY: John Wiley & Sons, 1998 – p.2).

Ці визначення збираються в результуючий трикутник «Час-Якість-Гроші» більш відоме як – «Швидко-Якісно-Дешево» – на цю тему є добре відомий жарт, суть якого зводиться до наступного: «Залиште тільки два пункти» ☺).

В процесі виконання проєкту вирішується завдання зберегти вартість у визначених межах, витримати терміни виконання за певного рівня якості. Завдання керування проєктом і полягає в тому, щоб урівноважити ці три складові : вартість, графік виконання проєкту і якісний результат. Але так зазвичай не відбувається. Графік, бю-

джет та якість не витримуються на потрібному рівні. Тому керівники проєкту змушені обирати у вигляді кінцевої мети лише один чи два параметри трикутника. Як вже зазначалося, – на професійному сленгу ця дія відома як «Якісно-швидко-дешево-вибери два з них».

Інститут управління проєктами (PMI) дає таке визначення проєкту:

> «***Проєкт*** *– це тимчасове зусилля, зроблене для того, щоб створити унікальний продукт або послугу з певною датою початку та закінчення дії, що відрізняється від продовження, повторних дій та потребує прогресивного вдосконалення характеристик*».

- **Мета.**

Має бути чітко визначена мета або ряд цілей проекту. По завершенню проєкту має бути отриманий якийсь результат. Якщо проєкт передбачає досягнення кількох цілей, то вони мають бути взаємопов'язані та несуперечливі.

- **Момент початку та завершення проєкту.**

Проєкт має протяжність у часі. У нього є чітко визначений початок і кінець дії, пов'язаний з часовою шкалою якихось визначених дат. Підтримка ПЗ не є проєктом і зазвичай являє собою пролонговану за межі проєкту дію та/але може бути внесена в проєкт(наприклад, як окремі версії).

- **Унікальність.**

Проєкт – одноразова сутність, що не повторюється при повторенні подібного по суті проєкту. Але повторювана робота теж може бути проєктом. Будівництво будинку зазвичай визначається як проєкт, незважаючи на те, що

підрядники побудували вже десятки будівель. Нехай зразок і процес в основному збігаються, виконуються за шаблоном, але є достатньо відмінностей у кожному будинку (це колектив, розміщення, матеріали тощо). Інакше йдеться про потокову лінію, де ідентичні частини виконуються подібним чином. Теж саме справедливо для професіоналів у галузі розробки ПЗ – вони ніколи не створюють ідентичну програмну систему, хоча можу її копіювати та складати з наявних рішень переносити довільним чином.

- **Обмеження.**

У проєкті є обмеження за вартістю, графіком розробки та якістю виконання. Ці обмеження формують трикутник, який для досягнення успіху має бути збалансований та керований.

(Project Management Institute. «A Guide to the Project Management Body of Knowledge» Sylva,
NC: PMI Publication Division. 1996.– p.167).

Отже, проєкт у термінах розробки ПЗ:

***Проект** – це унікальна, тимчасова дія з певними датами початку та кінця, спрямоване на те, щоб досягти однієї або кількох цілей при обмеженнях за вартістю, графіком та якістю виконання.*

Що таке керування проєктами?

Інститут управління проєктами (PMI) визначає ***керування проєктами*** як:

«набір перевірених принципів, методів та методик, застосованих для ефективного планування, створення графіка, керування та відстеження результатів роботи,

орієнтованої на успішне виконання, а також визначальних базис для планування проєктів».

Керцнер дає визначення ***керуванню проєктами*** як:

«планування, організацію, контроль та управління ресурсами компанії, виділеними в рамках певного проєкту. Ці ресурси призначаються для досягнення короткострокової мети, яка була встановлена для досягнення певних цілей та намірів».

Виокремимо загальні думки в цих визначеннях:

- **керування** – вміння в галузі управління проєктами – це частина загальних вмінь у сфері управління;
- **навички** – навички в галузі управління проєктами – це використання вмінь у сфері управління для досягнення запланованого результату. Вони включають і планування, і організацію виробничого процесу, і складання графіка, і контроль, управління та аналіз.

Отже, підсумовуючи, отримаємо визначення керування програмними проєктами.

***Керування програмними проєктами** – спеціалізація загального менеджменту, що визначає застосування керівних стандартних навичок планування, комплектування персоналом, організації, а також управління та контроль для досягнення наперед визначеної мети проєкту.*

Що таке командна розробка?

Були часи, коли програміст самостійно визначав, проєктував, писав та перевіряв свою роботу. Цьому сприяло оточення: проста, однокористувацька природа

використовуваних засобів, невеликі розміри додатків, що розробляються. Це дозволяло вважати розробку ПЗ індивідуальною діяльністю. Але, в певний момент, часи змінилися: «Розробка програмного забезпечення стала командним видом спорту» (*Г. Буч, 1998*).

Що ж таке «командна розробка», «команда» та «група»? Група може складатися з будь-якого числа людей, які взаємодіють між собою, психологічно приймають інших членів групи та відносяться до частини групи.

Команда – це група людей, які впливають один на одного заради досягнення спільної мети. Головна відмінність команди від групи – це результат ефективної взаємодії між людьми на основі загальних прагнень та цінностей, а також взаємодоповнюючих умінь (*skills*), що призводить до того що, загальні зусилля команди набагато перевищують суму зусиль її окремих членів. Як і в спорті, командна робота надзвичайно важлива для вирішення проблем конкурентоспроможності на загальному ринку, де індивідуальна майстерність менш важлива, ніж високий рівень колективних дій. У сучасному світі інформаційних технологій команда – це єдина умова виживання, а не виняток. Характерною рисою команди є широке коло повноважень або прав ухвалення рішення кожного члена команди.

Команда розробників програмного забезпечення повинна володіти наступними професійними навичками:

- вміння правильно розуміти проблему, вирішення якої покликане вирішити розроблюване програмне забезпечення (ПЗ);

- здатність виявлення вимог до розроблюваного ПЗ через спілкування з користувачем системи та іншими зацікавленими особами;
- вміння перетворити розуміння проблеми та потреб клієнтів у початкове визначення системи, яке задовольнятиме ці потреби;
- вміння керувати масштабом проєкту;
- вміння уточнювати визначення системи до рівня деталізації, придатного для проєктування та реалізації;
- здатність оцінити правильність розроблюваного ПЗ, проведення його верифікації та керування змінами.

Командна розробка – це процес розробки програмного забезпечення, який реалізує:

- керування діяльністю команди;
- керування завданнями окремого працівника та команди в цілому;
- вказування на компоненти, які слід реалізувати;
- надає критерії для відстеження та вимірювання продуктів, та функціонування проєкту.

Важливим фактом у командній розробці є те, що члени команди мають відмінності у професійних навичках та вміннях. Баланс та різноманітність навичок – це дві найважливіші складові відмінної команди. Саме це й робить команду командою. Одні «гравці» здатні ефективно працювати із замовниками, інші мають навички програмування, треті вміють тестувати програми, четверті – це фахівці з проєктування та архітектури систем.

Аналіз проблем одиночної та командної розробки програмного забезпечення

У теперішній час успішно використовуються одиночна та командна розробка ПЗ. Але сьогодні абсолютна більшість великих проєктів – результат колективної праці. У той же час «одинаки» – це й численні фрілансери, і програмісти, що поєднують роботу на компанію з одиночною розробкою.

Інженер компанії Sun Microsystems Ітан Ніколс, відчуваючи крайню скрутність у засобах існування (нещодавно став батьком), змушений був шукати джерела доходу і надихнувся прикладом успіху незалежного розробника iPhone-додатків Стіва Деметера. Ітан Ніколс розробив головоломку Trism, яка принесла йому близько $600 000 прибутку. Інший приклад програміста-одинака – автор «БітТорент», американський програміст Брем Коен.

Давайте спробуємо проаналізувати особливості одиночної та командної розробки ПЗ:

Одиночна розробка ПЗ	Командна розробка ПЗ
Необхідність самому знаходити замовників і вести з ними переговори (можливо англійською мовою) про обсяг, якість, термін та вартість робіт	Пошуком замовників і супроводом розробки професійно займаються спеціально навчені співробітники компанії
Прийняття всіх рішень (мова програмування, архітектура, дизайн програми та ін.) здійснюється однією особою, ґрунтуючись на попередньому досвіді певного розробника	Прийняття всіх рішень (мова програмування, архітектура, дизайн програми та ін.) здійснюється з урахуванням думки декількох розробників, ґрунтуючись на попередньому досвіді

Одиночна розробка ПЗ	Командна розробка ПЗ
Замовник безпосередньо спілкується з розробником, що унеможливлює ефект «зіпсованого телефону»	Замовник не завжди спілкується з розробником напряму, що може створити ефект «зіпсованого телефону»
Розробник вільний у виборі замовника, часу та місця роботи	Розробник обмежений роботою в проєктах компанії згідно зі штатним розкладом компанії
Розробник більш схильний до ризиків несплати (неповної оплати) виконаної роботи	Дохід розробника за виконану роботу, як правило, гарантований компанією

Аналіз термінів предметної галузі

Ми проаналізуємо наступні терміни: процес, проєкт, персонал, продукт, якість.

Те, як ми створюємо ПЗ, яку послідовність дій при цьому виконуємо, як взаємодіємо з іншими членами команди, яких норм та правил дотримуємось в роботі, як наш проєкт взаємодіє із зовнішнім світом – це й називається процесом розробки ПЗ. Основою успішності розробки ПЗ є методологічно правильна побудова процесу розробки, його усвідомлення та поліпшення.

Процес створення ПЗ представлений безліччю різних видів діяльності, методів і методик, таких, як: розробка технічного завдання, архітектури застосування, кодування, тестування, написання документації та ін.

Нині не існує універсального процесу розробки ПЗ, дотримуючись якого ми можемо гарантовано отримати якісний продукт в обумовлений термін. Кожна конкретна розробка, що здійснюється певною командою розробників, має велику кількість індивідуальних особливостей.

Але все ж таки перед початком проєкту необхідно спланувати процес роботи, визначивши ролі та обов'язки у команді, плани та терміни виконання проміжних та остаточної версій продукту.

Результатом **процесу розробки** програмного забезпечення та підсумком проєкту є **продукт**, готовий до постачання (**кінцевий продукт**). До нього входять: тіло – початковий код, у вигляді компонентів, які можуть бути відкомпільовані та виконані, посібник з експлуатації та додаткові складові частини поставки. Перш ніж продукт стає готовим до постачання, він проходить стадію робочого продукту.

Робочий продукт – це проміжні результати процесу розробки ПЗ, які допомагають ідентифікувати, планувати та оцінювати різні частини результату. Проміжні результати допомагають менеджерам різних рівнів відслідковувати процес реалізації проєкту, а замовник отримує можливість ознайомитися з результатами задовго до закінчення проєкту. Розробники проєкту у своїй щоденній роботі отримують простий та ефективний спосіб обміну робочою інформацією – обмін результатами.

Розробка ПЗ відноситься до проєктних видів діяльності. Проєкти поділяються на промислові та творчі, відповідно до різних принципів управління. Так от, розробку ПЗ відносять до творчих проєктів.

Промислові проєкти часто поєднують велику кількість різних організацій за невисокої унікальності самих робіт, наприклад – будівництво багатоповерхового будинку. До них можна віднести різні міжнародні проєкти, і не лише промислові, а й освітні, культурні та ін. Головне завдання

в управлінні такими проектами – все передбачити, всіх проконтролювати, нічого не забути, все зуміти зібрати «до купи», досягти узгодженої взаємодії.

Творчі проєкти характеризуються абсолютною новизною ідеї – новий вид обслуговування, повністю новий програмний продукт, який не має аналогів на ринку, сюди також можна віднести проєкти в галузі мистецтва та науки. Кожен бізнес на початку, як правило, стає таким от «творчим» проєктом. Новизна в подібних проєктах не абсолютна – таке може, вже й було, але для команди проєкту – вперше. Тут йдеться про величезний обсяг новизни для самих людей, які втілюють цей проєкт. Проєкти з розробки програмного забезпечення вбирають частини промислових проєктів та, безсумнівно, проєктів творчих, перебуваючи між цими полюсами. Часто вони складні тому, що об'ємні і знаходяться на стику різних дисциплін – цільового бізнесу, де має застосовуватися програмний продукт, і складного, нетривіального програмування. Часто у програмні проєкти додається розробка унікального обладнання. З іншого боку, оскільки програмування активно проникає в різні сфери людської діяльності, то відбувається створення абсолютно нових, унікальних програмних продуктів, і їх розробка та просування мають усі риси творчих проєктів.

На етапі розробки проводяться регулярні вимірювання розробленого продукту для встановлення відповідності вимогам. Будь-які невідповідності мають розглядатися як дефекти – відсутність якості.

Якість продукту визначається в стандарті ISO 9000:2005, як ступінь відповідності його характеристик

до вимог. Якісного програмного продукту не створити без якісного процесу розробки.

Методами забезпечення якості ПЗ є:

1. Створення та вдосконалення якісного процесу розробки ПЗ.
2. Забезпечення якості коду шляхом дотримання стандартів оформлення коду в проєкті та контроль за дотриманням цих стандартів. Сюди входять правила створення ідентифікаторів змінних, методів та імен класів, на оформлення коментарів тощо. Якість коду забезпечується також рефакторингом – переписуванням коду, але не з метою додавання нової функціональності, а для покращення його структури.
3. Тестування – найпоширеніший метод, без якого сьогодні не випускається жоден продукт.

Якщо ПЗ створюється командними зусиллями, то висока якість і продуктивність з'являються, коли **персонал**, тобто члени команди ефективно залучені до спільної справи і зберігають мотивацію, а вся команда діє ефективно. Для управління командою потрібно перейти до сфери відносин між людьми. Зазвичай використовується ієрархічна структура команди. Очолює команду менеджер проєкту, який підпорядковується лідеру напряму (начальнику відділу). Звичайна команда складається з розробників, адміністратора баз даних, тестерів ПЗ та можливо інших фахівців. Великий проєкт може вміщувати лідерів модулів, кожен з яких має у своєму підпорядкуванні декількох розробників.

Мета менеджера проєкту полягає в тому, щоб отримати впевнену у своїх силах команду. При цьому необхідно враховувати такі людські фактори:

- кваліфікацію, підготовку та досвід членів команди;
- особисті прагнення та кар'єрне зростання членів команди;
- потреби у наставництві та розвитку.

Дуже важливим моментом є спілкування в команді. Команда, члени якої спільно працюватимуть кілька місяців задля досягнення спільної мети, повинна становити єдине ціле, між її членами має бути повне порозуміння.

Для підтримки проінформованості членів команди про просування проєкту та його проблеми менеджери проєктів можуть користуватися такими методами:

- організація дошок оголошень для повідомлень та звітів про проєкт;
- електронні розсилки по проєкту;
- internet (intranet) – сайт для публікації документів, пов'язаних з проєктом;
- наради щодо проекту.

Підбиваючи підсумки, дамо визначення термінам.

Проєкт *– організаційна сутність, яка використовується для керування розробкою ПЗ. В результаті проєкту з'являється програмний продукт.*

Персонал *– розробники, архітектори, тестери, керівники, користувачі, замовники та всі інші зацікавлені особи – це основний двигун програмного проєкту.*

Якість *– характеристика ПЗ як ступеня його відповідності до вимог. Відповідність вимогам передбачає, що вимоги мають бути настільки чітко визначені, що вони не можуть бути зрозумілі та інтерпретовані некоректно.*

Продукт *– артефакти (від латинського artefactum – дещо штучно виготовлене), створювані в процесі діяльності проєкту. До них відносяться моделі, тексти програм, виконувані файли і документація.*

Процес створення ПЗ *– це визначення повного набору видів діяльності, необхідних для перетворення вимог користувача на продукт.*

Характеристики проєкту

З характеристик проєкту можна виділити наступні:

- **тип проєкту;**
- **мета проєкту;**
- **вимоги до якості;**
- **вимоги до бюджету;**
- **вимоги щодо термінів завершення.**

Проєкт з розробки ПЗ має чотири характерні ознаки:

1. Спрямованість на досягнення конкретних цілей.
2. Координоване виконання взаємопов’язаних дій.
3. Обмежена довжина в часі з певним початком та кінцем, та обмеження у використовуваних ресурсах (якісно-швидко-дешево).
4. Неповторність та унікальність (хоча б часткова).

Це і є ознаки, що відрізняють проєкти від інших видів людської діяльності.

Проєкти спрямовані на досягнення конкретних цілей

Усі зусилля з планування та реалізації проєкту робляться для того, щоб його цілі досягалися і отримувалися конкретні результати. Саме цілі є рушійною силою проєкту.

Тому важливо при керуванні проєктом точно визначити та сформулювати цілі, починаючи з верхнього рівня і до найдрібнішої деталізації цілей та завдань.

Координоване виконання взаємопов'язаних дій

Проекти завжди мають складові частини та персонал, який реалізує певні завдання. Деякі частини проєкту є залежними, дії персоналу не бувають ізольованими. Для їх правильного об'єднання та отримання підсумкового результату необхідно організувати їхню взаємодію. Порушення синхронізації виконання різних частин ставить під загрозу виконання проєкту. Проєкт – це динамічна система, що складається із взаємозалежних елементів і потребує особливих підходів до управління.

Обмеження коштів

Грошей завжди недостатньо. Часу завжди мало. Крім того, проєкт обмежений рамками законів держави. Необхідно пам'ятати про персонал, який має якісь моральні принципи, що також може бути свого роду обмеженням. І врешті-решт, проєкт обмежений законами природи. Ось основні обмеження проєкту.

Неповторність та унікальність

Через постійну зміну умов реалізації проєкту, планування має бути гнучким настільки, що можуть змінюватися навіть початкові цілі та методи їх досягнення. Неможливо передбачити заздалегідь усі зміни, які супроводжуватимуть виконання проєкту.

Закон Лермана наголошує: «*Будь-яку технічну проблему можна подолати, маючи достатньо часу та грошей*». Закон має наслідок *(Лерман)*: «*Вам ніколи не вистачатиме або часу, або грошей*». Методика управління діяльністю на основі проекту була розроблена саме для подолання сформульованої проблеми Лермана. Зазвичай керівник проєкту розуміє своє основне завдання у виконанні проєкту як забезпечення виконання робіт.

Більш досвідчений керівник проєкту точніше сформулює визначення головного завдання менеджера проєкту: «*Забезпечити виконання робіт у строк, у межах виділених коштів та відповідно до технічного завдання*». Саме ці три складові: час, бюджет та якість робіт знаходяться під постійною увагою керівника проєкту. Тому ще розширимо визначення проєкту:

> ***Проєктом** називається діяльність, спрямована на досягнення певних цілей з максимально можливою ефективністю при заданих обмеженнях за часом, коштам та якості кінцевого продукту.*

Для того щоб впоратися з обмеженнями часу, використовуються методи побудови та контролю графіків робіт. Для управління грошовими обмеженнями вико-

ристовуються методи формування фінансового плану (бюджету) проєкту і, в міру виконання робіт, дотримання бюджету відстежується, щоб не дати витратам вийти з-під контролю.

Приклад:

№ етапу робіт	Опис етапу робіт	Складність етапу робіт	Трудомісткість етапу робіт (беручи за основу дані з попередніх	Трудомісткість етапу робіт, факт	Вартість етапу робіт, план	Вартість етапу робіт, факт
1	Оновлення персональних даних	висока	8 людино-годин			
2	Додати адресу	середня	6 людино-годин			
3	Оновлення адреси	середня	6 людино-годин			
4	Видалення адреси	низька	4 людино-години			
5	Додати телефонний номер	середня	6 людино-годин			
	...					
	РАЗОМ:					

Розробляти програмне забезпечення дуже непросто, а розробляти якісне програмне забезпечення і при цьому завершувати роботу вчасно – ще складніше.

Згідно з Jet Info Online, наприкінці 60-х років минулого століття, в США відбулася криза ПЗ (*Software Crisis*). Сюди входило відставання від графіка, перевищення кошторису у великих проєктах.

Крім того, розроблене ПЗ не мало контрактних функціональних можливостей, мало низьку продуктивність і його якість не задовольняла споживачів.

За результатами досліджень, зроблених у 1995 році компанією Standish Group, що проаналізувала роботу 364-х американських компаній, результати виконання понад 23-х тисяч проєктів, пов'язаних з розробкою ПЗ, виглядали наступним чином:

- лише 16,2% завершилися вчасно, реалізували всі необхідні функції і можливості, і не перевищили запланований бюджет;
- 52,7% проєктів завершилися із запізненням, необхідних функцій не було реалізовано в повному обсязі, витрати перевищили запланований бюджет;
- 31,1% проєктів було припинено до завершення;
- для двох останніх категорій проєктів термін виконання середнього проєкту виявився перевищеним на 122%, а бюджет – на 89%.

У 1998 році відсоткове співвідношення трьох зазначених категорій проєктів лише трохи змінилося на краще (26%, 46% та 28% відповідно).

За оцінками провідних аналітиків, незначна зміна на краще процентне співвідношення трьох вказаних категорій проєктів в останні роки відбувається, здебільшого не за рахунок підвищення керованості та якості проєктування, а за рахунок зниження масштабу виконуваних проєктів.

Серед причин цих невдач, на думку експертів, можна вказати наступне:

- відсутність постійного контакту із замовником;

- недостатньо чітке та повне формулювання вимог до ПЗ;
- нестача необхідних ресурсів;
- бездарне планування та неграмотне керування проєктом;
- часте зміна вимог, специфікацій, умов;
- недосконаліть, недостатньо знань або новизна використовуваної технології;
- відсутня підтримка з боку вищого керівництва;
- низька кваліфікація розробників, брак необхідного досвіду.

Витрати, пов'язані з проєктом

Для розробки ПЗ, звісно, потрібні капіталовкладення. На що вони витрачаються?

На такі статті витрат:

- зарплата розробникам;
- оренда приміщення;
- амортизація засобів розробки (комп'ютери, мережі, програмні продукти);
- оплата енергоресурсів;
- оплата керуючого персоналу.

Відповідно до існуючої термінології, витрати поділяються на прямі та непрямі.

До **прямих витрат** належать витрати, які безпосередньо переносяться на собівартість конкретної продукції або послуг. Сюди входять: матеріали, заробітна плата

основних виробників (програмісти, тестувальники, керівники відділів та частин проєкту, наукові співробітники, консультанти і т. д.), відрахування на соціальні потреби (пенсійний фонд, фонд соціального страхування, фонд зайнятості та фонд обов'язкового медичного страхування), електроенергія, витрати за минулий період, віднесені на вартість продукції звітного періоду та інші прямі витрати. Ці витрати і непрямі витрати характеризують групу витрат виробництва за способом включення їх у собівартість продукції: вони безпосередньо покладаються на рахунок виробництва.

Непрямі витрати – це витрати, які важко пов'язати з якоюсь конкретною діяльністю або проєктом, але, тим не менш, необхідні для нормального функціонування організації та успішного виконання її завдань: оплата праці адміністрації, реклама, вартість зносу основних фондів, амортизації капітального обладнання, загальні комунальні витрати (телефон, газ, електрика, ліфт, антена та ін.).

2. МОДЕЛІ ТА МЕТОДОЛОГІЯ ПРОЦЕСУ РОЗРОБКИ

Загальний огляд моделей та методологій процесу розробки

Історично склалося кілька моделей процесу розробки програмних систем. Перша, яку називають «класичною», прийшла з інженерної практики та відповідає процесам, що прийшли з конструювання систем. Усі роботи, що становлять класичну модель, виконуються послідовно, один за одним у заздалегідь визначеному порядку. На жаль така добре зрозуміла модель не працює майже ніколи – в процесі розробки постійно виникають нові вимоги та уточнення, відмова від яких призводить до конфлікту із замовником. Розподіл процесів кодування та тестування призводить до важко здійснюваних переробок коду. Пізніше виникли нові види – ітераційні, що дозволяють проводити коригування та уточнення багаторазово, і включають тестування в процес розробки від її початку.

Фази процесу

У будь-якому випадку, незалежно від моделі, процес розробки можна розподілити на декілька окремих складових частин, які називаються **фазами процесу проєктування**. Саме послідовність їх виконання й визначає ту чи іншу модель (див. рис. 3).

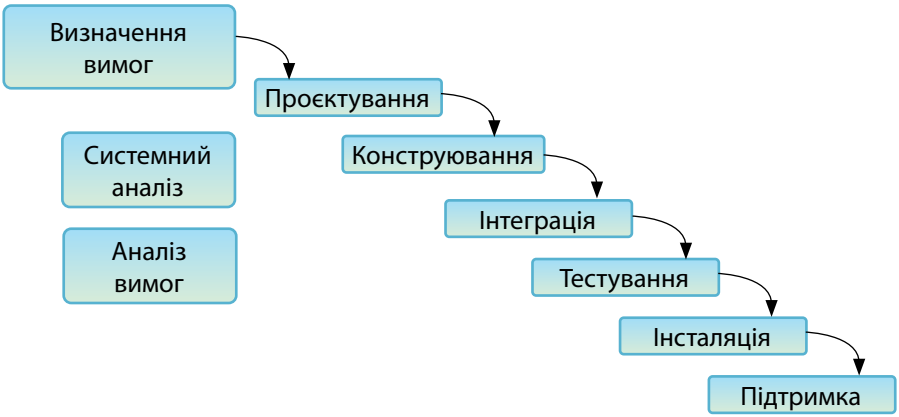


Рисунок 3. Класична модель. Фази процесу розробки

Фази процесу розробки

1. **Визначення вимог.**

Ця фаза необхідна для з'ясування, чого хоче замовник і що зробити, щоб це функціонувало. Система, що розробляється, розглядається як одне ціле, визначається функціональність системи, характеристики обладнання, на якому система функціонуватиме, фізичне розміщення системи, перше наближення стосовно визначення витрат на розробку. Зазвичай ця фаза поділяється на дві складові: системний аналіз та аналіз вимог.

Системний аналіз – ця частина фази визначення вимог задає роль кожного елемента в комп'ютерній системі та описує взаємодію елементів один з одним. Він починається з визначення вимог до кожного елемента системи та об'єднання цих вимог у програмну частину системи. Коли формується інтерфейс ПЗ з іншими складовими частинами системи – апаратурою, людьми, базами

даних тощо, найяскравіше проявляється необхідність системного підходу. У той же час вирішується завдання планування проєкту ПЗ, у ході якого визначаються обсяг та ризик кожної з проєктних робіт, розраховуються необхідні трудові витрати, оформлюються робочі завдання та план-графік виконання робіт.

Аналіз вимог – ця частина фази визначення вимог відноситься до програмної складової – програмному забезпеченню. При аналізі вимог проводиться уточнення та деталізація функцій ПЗ, його характеристик та інтерфейсу. Усі зроблені визначення зберігаються у документі під назвою «специфікація аналізу». Ця частина фази визначення вимог завершує планування проєкту.

2. **Проєктування.**

Фаза проєктування визначає те, що стане основою розробки, дозволить відобразити:

- архітектуру ПЗ;
- модульний склад ПЗ;
- алгоритмічну структуру ПЗ;
- структури даних;
- вхідний та вихідний інтерфейси (форми введення та звіти).

Початкові дані для проєктування записуються у вигляді ***специфікації аналізу***. В ході проєктування, вимоги до ПЗ трансформуються у безліч проєктних представлень (наприклад, безліч UML-діаграм). Під час проєктування ПЗ, основна увага приділяється якості створюваного

програмного продукту. Важливо правильно визначити підсистеми ПЗ та способи їх взаємодії.

3. **Конструювання («реалізація», «кодування»).**

Зазвичай саме цю фазу називають програмуванням, хоча вона вирішує лише роль перекладу того, що ми вже знаємо, якоюсь мовою програмування. Усі підсистеми, визначені на етапі проєктування, розробляються паралельно (якщо вистачає ресурсів: фірма з однієї людини не може розробляти підсистеми паралельно ☺).

4. **Інтеграція.**

Якщо інтерфейси підсистем розроблені відповідно до специфікації, підсистеми не надають побічного впливу одна на одну, і при розробці специфікації враховані усі вимоги – процес складання або об'єднання підсистем не викликає жодних труднощів. Такого ніколи не буває. По-перше, не усі підсистеми розробляються одночасно. По-друге, помилки в одній підсистемі часто призводять до неправильного функціонування групи залежних підсистем. Тому, під час збору використовують два підходи: метод «великого вибуху» (*зараз усе запрацює!*) та метод послідовного підключення. Ні той, ні інший не дозволяють з абсолютною точністю визначити підсистему, яка веде до помилок. Під час «великого вибуху» усі підсистеми вмикаються одночасно, кількість помилок зашкалює і зазвичай намагається вирішити проблему шляхом послідовного підключення. Натомість є підсистеми, які залежать від ще непідключених підсистем настільки, що при ввімкненні перестають виконувати свої функції і визначити причину складно. Тому починається наступна фаза.

5. **Тестування та налагодження («верифікація»).**

На етапі тестування виконується пошук дефектів. Для спрощення пошуку дефектів, розробляються тестові приклади. Успішне або неуспішне виконання прикладу називається тестовим випадком. Теоретично неможливо перевірити усі гілки програмного коду, тому насправді кількість тестових прикладів та час на тестування обмежено. Тестування поділяється на альфа-тестування (сам розробник) та бета-тестування (замовник або треті особи, які не є ні замовником, ні розробником). Знайдені дефекти вирушають на доопрацювання.

6. **Інсталяція.**

Ця фаза виявляє недоліки та проблеми, пов'язані з роботою розроблюваного ПЗ на обладнанні та в оточенні замовника.

Можливі різні проблеми. Назвемо деякі з них:

- в програмі жорстко забитий шлях, якого немає на комп'ютері замовника (або хост);
- в програмі використовується служба, яка відсутня в мережі замовника (наприклад, ви приносите програму на C#, а в замовника відсутня або неактуальна версія .Net Framework);
- у замовника комп'ютер та операційна система відповідають вимогам специфікації, але через перевантаження комп'ютера різними фоновими процесами ваше програмне забезпечення не в змозі підтримувати потрібні характеристики інтерактивної роботи (*випадок з практики автора*).

7. **Підтримка.**

В процесі експлуатації замовником розробленого ПЗ можуть виникати різні ситуації, які можуть бути вирішені тільки разом із розробником:

- змінили обладнання (принтер, монітор, мишка, якийсь адаптер, жорсткий диск, процесор і т. п.);
- змінили вимоги до генерованих звітів (нові форми податкової) або інтерфейсу («*приберіть пікалку*»);
- необхідність оновлення базових довідників, які можуть бути скориговані тільки розробником.

Усі ці питання, не вирішені своєчасно, завдають шкоди престижу фірми та кількості можливих замовників (антиреклама). Тому виконавець вносить у план розробки фазу підтримки. Підтримка може виконуватися не самим виконавцем, а довіреною особою (наприклад, добре відома система 1С).

Крім описаних фаз можлива наявність й інших, але ці 7 присутні практично в усіх моделях розробки.

Ще раз перелічимо фази:

1. Визначення вимог.
2. Поєктування.
3. Конструювання.
4. Інтеграція.
5. Тестування.
6. Інсталяція.
7. Підтримка.

Наявність та взаємне розташування цих фаз, а також додавання деяких нових фаз або функцій, покладених на

фазу, взаємодія колективу в процесі виконання кожної фази – все це визначає так звану **модель розробки**.

Спробуємо оцінити переваги та недоліки найпоширеніших моделей.

Водоспадна модель

Ще її називають **каскадною** або **класичною**. Автор **Вінстон Ройс**. Представляє строгу і зрозумілу побудову. Немає повернення на попередню фазу. Замовник бачить готовий проєкт тільки в кінці проєктування, коли у замовника може змінитися уявлення про природу ПЗ, який він купує.

На рисунку 4 видно, що вона відрізняється від попередніх іншою назвою фаз, деякі фази зібрані в одну, а деякі – розбиті на складові.

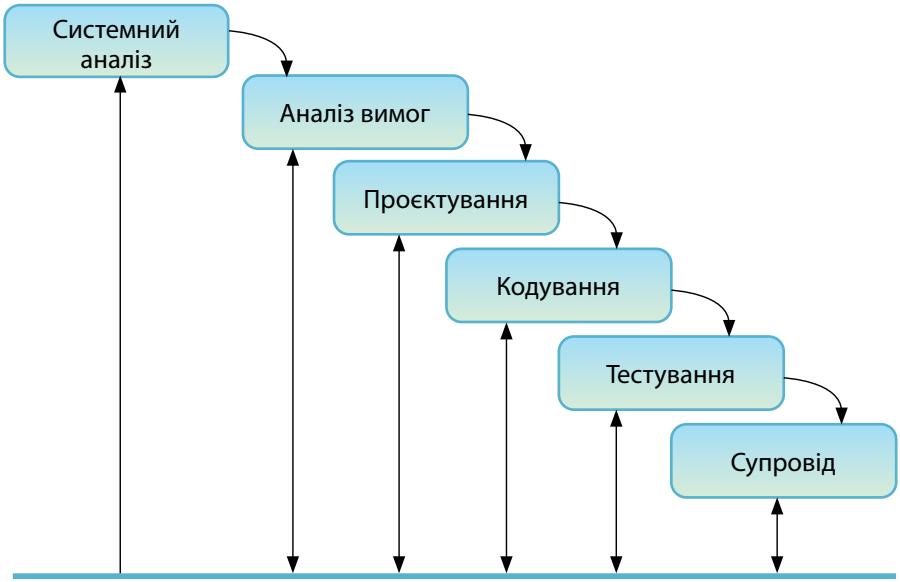


Рисунок 4. Каскадна модель Вінстона Ройса

Це застаріла, але все ще часто застосована на практиці модель. Вона надає план та тимчасовий графік для всіх етапів проєкту, впорядковує хід конструювання. Потрібно наголосити, що реальні проєкти часто потребують зміни стандартної послідовності кроків. Тим не менш, завдяки тому, що життєвий цикл розробки заснований на точному формулюванні початкових вимог до ПЗ, більшість замовників визнає саме цю модель, хоча на практиці, на початку проєкту, замовник може визначити свої вимоги лише частково. Така модель має найбільшу кількість незавершених проєктів.

Макетування

Макетування – це не модель, а методологія роботи з замовником, яка успішно використовується в різних моделях проєктів.

Якщо замовник не може одразу сформулювати вимоги для ПЗ, що розробляється, або розробник не впевнений у нормальному функціонуванні ПЗ у реальному середовищі замовника, в реалізації діалогу з користувачем або в ефективності реалізованого алгоритму заведено використовувати **макетування**.

Основна мета макетування – позбутися від невизначеностей у вимогах замовника (рис. 5).

Макетування (*ще його називають* ***прототипуванням***) – використання спрощеної моделі замість потрібного програмного продукту.

Така модель приймає одну із трьох форм:

- паперовий макет або макет на основі ПК (зображується або малюється людино-машинний діалог);

- працюючий макет (виконується деяка частина необхідних функцій);
- існуюча програма (характеристики якої потім мають бути змінені).

Макетування ґрунтується на багаторазовому повторенні ітерацій створення все більш точного макета, в яких беруть участь і замовник, і розробник.

В принципі, макетування може застосовуватися в будь-яких моделях розробки ПЗ як складова частина фази визначення вимог.

Переваги макетування: макетування забезпечує найповніше визначення вимог до ПЗ.

Недоліки:

- замовник може вирішити, що макет – це розроблювана система;
- розробник може вирішити, що макет – це розроблювана система.

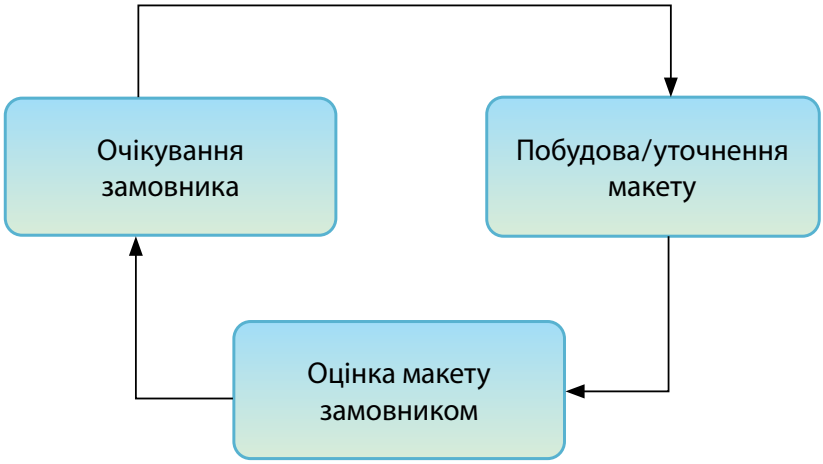


Рисунок 5. Загальна модель роботи з макетом

Спіральна модель

Спіральну модель запропонував **Баррі Боем** у 1988 році. В основу спіральної моделі покладені:

- макетування;
- сильнозмінена водоспадна модель, закручена в еволюційну спіраль;
- нова складова частина – аналіз ризиків.

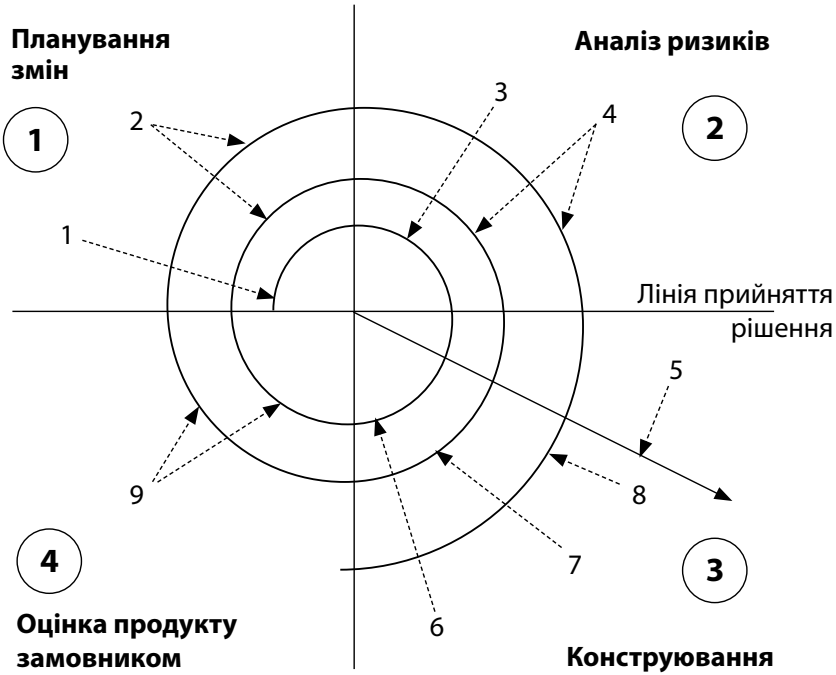


Рисунок 6. Спіральна модель

Спіральна модель (рис. 6):

1 – первинний збір вимог та планування проєкту; 2 – облік рекомендацій замовника щодо внесення змін; 3 – аналіз ризику на основі первинних вимог; 4 – аналіз ризику на основі реакції замовника на попередній етап; 5 – перехід до комплексної системи; 6 – початковий макет системи; 7 – наступний рівень макета системи; 8 – чергова реалізація; 9 – оцінка реалізації замовником.

До спіральної моделі входить чотири етапи, розміщені в чотири квадранти по спіралі (див. рис. 6):

1. **Планування** – визначення цілей проєкту або чергового «витку» розробки, можливих варіантів рішень та усіляких обмежень.
2. **Аналіз ризику**: аналіз варіантів та вибір – виконувати подальшу розробку або ризик перевищує можливий позитивний ефект.
3. **Конструювання** – перехід до наступного рівня розробки.
4. **Оцінювання** – тестування та оцінка замовником поточних результатів розробки.

Спіральна модель відноситься до еволюційних моделей, характерна риса яких – багаторазовий «міні» життєвий цикл, з виходом по закінченні на новий рівень, новий «виток спіралі», зі збільшенням кількості доступних замовнику функцій та зростанням якості продукту. Замовник досить часто залучається до процесу розробки та після кожного циклу може бачити та оцінити стан системи.

Переваги:

- близько до реальності в еволюційному вигляді відображає процес розроблення програмного забезпечення;
- явно враховує ризики на кожному еволюційному «витку» розробки;
- включає до ітераційної моделі розробки системний підхід (водоспадну модель);
- використовує макетування для зниження ризику та вдосконалення програмного виробу.

Недоліки:

- немає достатньої статистики ефективності моделі;
- підвищуються вимоги до замовника;
- виникають проблеми з контролем та керуванням часом розробки (замовнику не видно, коли ж буде готовий кінцевий, розробник не знає, чи забезпечений він на майбутній період).

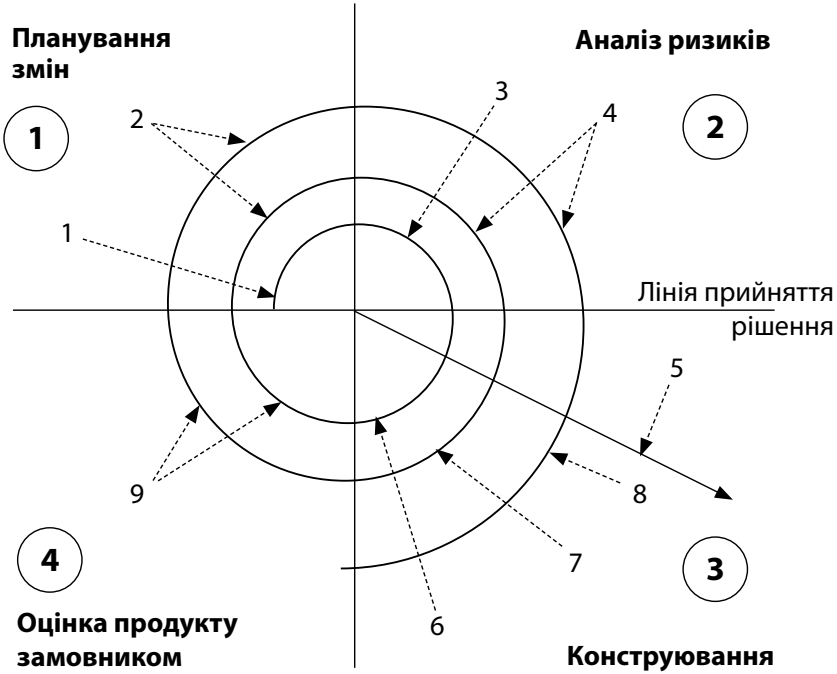
Компонентно-орієнтована модель

Компонентно-орієнтована модель – це розвиток спіральної моделі. Вона також ґрунтується на еволюційній стратегії конструювання. Тут уточнюється зміст квадранта конструювання: відображається той факт, що в сучасних умовах нова розробка має ґрунтуватися на повторному використанні існуючих програмних компонентів (див. рис. 7).

Ітеративна модель

В принципі, спіральна модель та всі еволюційні моделі є різновидом ітеративної моделі. Ітеративна модель вказує на повторення групи дій та вносить в процес розробки аналіз ризику після кожної ітерації. Крім спіральної моделі існує безліч методик, суть яких зводиться до мінімізації ризику за рахунок виконання у вигляді множини коротких циклів.

Назвемо деякі з них.



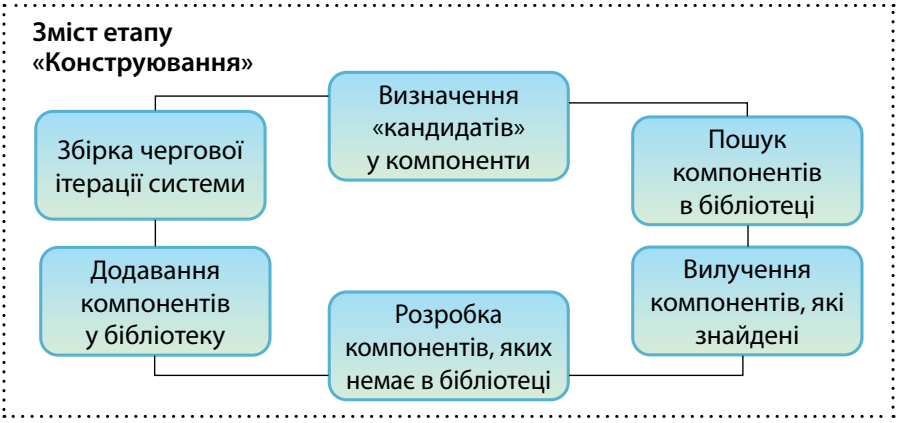


Рисунок 7. Компонентно-орієнтована модель

Agile

Agile – сімейство методів розробки у розробці програмного забезпечення, яке почалося зі спіральної моделі (1985 р.) – багаторазово повторюваний короткий, але повний цикл розробки. Наступним етапом розвитку Agile (еджайл) стала методологія Scrum, що спирається на 30-денні цикли, самоорганізацію команд розробників та щоденні мітинги (1990-1999 рр.). Ще один вид самоадаптованого ітераційного процесу розробки був запропонований авторами **RAD (Rapid Application Development)** у 1994-му. До 96-го року відносять опис методології **Кента Бека XP (eXtreme Programming)**, варіант ітераційного процесу розробки з гіпертрофованим тестуванням та передачею коду між членами команди (спільне володіння кодом). І, нарешті, остання з ітеративних методик, що увійшли в Agile – **Feature Driven Development (FDD)**, запропонована у 1997 році і одразу врятувала безнадійний проєкт.

Автори зазначених методологій опублікували їх узагальнені риси в ***Agile Manifesto*** – основному документі для процесів гнучкої, що допускає зміни вимог і навіть цілей розробки. ***Agile Manifesto*** визначає цінності та принципи, які застосовують найбільш успішні команди.

Підхід включає цінності, значимість яких у неуспішних командах зведено до нуля:

- процеси та інструменти не такі важливі, як особистості та їх взаємодії;
- працююче програмне забезпечення важливіше, ніж повна документація;

- реакція на зміни більш важлива, ніж неухильне виконання плану. Співпраця із замовником більш важлива, ніж зобов'язання за контрактом.

Ось принципи, які входять до ***Agile Manifesto:***

- щоденне спілкування замовника з розробником протягом реалізації проєкту;
- вітається зміни вимог навіть наприкінці розробки, що збільшує конкурентоспроможність отриманого продукту;
- часте постачання робочого ПЗ замовнику (можливо, щомісяця або тиждень);
- найкраще задоволення клієнта за рахунок більш ранньої і безперебійної поставки цінного ПЗ;
- усі діючі особи проєкту – спонсори, розробники та користувачі – повинні мати можливість підтримувати постійний темп на тривалий термін;
- простота розробки – виконувати розробку тільки справді необхідних частин;
- проєктом повинні займатися мотивовані особи, яких необхідно забезпечити потрібними умовами роботи, підтримкою та довірою;
- найбільш рекомендований спосіб спілкування – особиста розмова (віч-на-віч);
- працююче ПЗ – найкращий вимірювач прогресу розробки;
- постійна увага приділяється покращенню технічної майстерності та зручності дизайну;

- часта або постійна адаптація (поліпшення ефективності роботи) до змінюваних умов;
- у самоорганізованої команди виходять найкращі архітектура, вимоги та дизайн.

Далі розглянемо деякі Agile-методології більш детально.

Scrum

Scrum ***(скрам)*** – методологія керування проєктами для гнучкої розробки програмного забезпечення. Автори – **Джеф Сазерленд** та **Кен Швабер**. Методологія заснована на аналізі практик японських промислових компаній та щоденних мітингах фірми «Борланд».

Процес розробки поділяється на 30-денні цикли (***спринти***). На початку процесу розробки визначаються усі вимоги та цілі, та обираються пріоритети для першого циклу. В кінці кожного циклу команда на нараді (***мітингу, рев'ю***) надає керівництву та замовнику готовий до використання продукт, в якому поки що враховані не всі вимоги замовника, а тільки та функціональність, яка була запланована на цей спринт. Замовник разом з менеджером проєкту розглядають поточний стан продукту, оцінюють його. За результатами мітингу приймаюча сторона може зробити висновки про те, як розвивається система і що можна покращити, також вибирає, що потрібно зробити в наступному спринті.

Керівництво не втручається в роботу команди протягом усього циклу (***повна довіра***) і не ставить команді додаткових завдань після ствердження завдань на спринт. Для налагодження взаємодії команди та неухильного

дотримання правил скраму вибирається скрам-майстер. Команда щоденно проводить мітинги-звіти по кожному члену команди і вирішує перерозподіл зусиль для своєчасного вирішення поставленого кола завдань, що вважається глибоким зворотним зв'язком і дозволяє якісно контролювати процес розробки.

З погляду методології, Scrum містить набір методів, попередньо визначених ролей та артефактів (в даному випадку – контрольних точок, що визначають напрям розвитку, наприклад: артефактом може бути доведена незастосовність одного з підходів). Отримані артефакти обговорюються на мітингах.

Після закінчення спринту проводиться ***демонстрація*** та ***ретроспектива*** (спільний мітинг команди, керівництва та замовника).

Детальніше про методологію Scrum ви дізнаєтеся у наступних уроках.

Термінологія Scrum

Ролі:

- Власник Продукту (**Product Owner**)
- Керівник (**Scrum Master**)
- Команда (**Scrum Team**)
- Користувачі (**Users**)
- Клієнти, Продавці (**Stakeholders**)
- Експерти-консультанти **(Consulting Experts**)

Методи:

- Планування спринту (**Planning Meeting**)

- Зупинка спринту (**Sprint Abnormal Termination**)
- Мітинг (**Daily Scrum**)
- Демонстрація (**Demo Meeting**)
- Ретроспектива (**Retrospective Meeting**)

Артефакти:

- Product Backlog
- Sprint Backlog

XP

Впізнаєте? Та це ж застаріла версія Windows! ☺

Звісно, що ні! Абревіатура виникла від **eXtreme Programming.** Вся команда програмістів сідає в гамак або стає на лижі, можна – скейти, і... Ні.

Ідея екстремального програмування – доведення всіх складових гнучкого (***Agile***) програмування до абсурду. Якщо цикл розробки має бути коротким, то вибираємо якомога коротший цикл. Якщо зустрічі із замовником мають проводитися часто – давайте включимо представника замовника до команди. Якщо тестування потрібно проводити якомога частіше – давайте розробляти функцію продукту тільки після розробки тесту для неї. Якщо не слід перевантажувати продукт зайвим функціоналом – давайте визначати через замовника лише ті функції, які йому вкрай потрібні. Якщо замовник сумнівається в необхідності подальшої розробки – давайте відразу ж припиняти її. Якщо є труднощі в оволодінні програмістом чужого коду – зробимо так, щоб код розроблявся не одним, а одразу двома програмістами (парне програ-

мування, зміна програмістів у парах протягом проєкту). Якщо людина – не кінь і ми хочемо максимальної продуктивності та зосередженості протягом усього процесу розробки – 40-годинний робочий тиждень.

RUP

Rational Unified Process (RUP) – це метод розробки програмного забезпечення, створений компанією Rational Software, саме ця компанія забезпечила свій метод повним набором інструментів.

Перш за все, RUP – це ітеративна модель розробки. Наприкінці кожної ітерації (від 2 до 6 тижнів) розробники повинні досягти запланованих на цю ітерацію цілей, створити або доробити проєктні артефакти і отримати проміжну, функціональну версію програмного забезпечення.

Такий спосіб розробки дозволяє швидко реагувати на змінені вимоги, виявити та усунути ризик на ранніх стадіях проєкту, а також високоефективно контролювати якість створюваного продукту.

Ітерація розробки поділяється на 4 фази:

1. Початок (**Inception**).
2. Проєктування (**Elaboration**).
3. Побудова (**Construction**).
4. Впровадження (**Transition**).

Кожна ітерація завершується контрольною точкою, яка дозволяє оцінити успішність цієї ітерації.

В процесі розробки контролюється безліч процесів (для кожного є відповідний інтерфейс):

- моделювання бізнес-процесів;
- керування вимогами;
- аналіз та проєктування;
- реалізація;
- тестування;
- розгортання;
- конфігураційне керування та керування змінами;
- керування проєктом;
- керування середовищем.

MSF

Microsoft Solutions Framework (MSF) – це методологія розробки програмного забезпечення від Microsoft, фактично, варіант спіральної моделі.

Спіраль поділена на етапи контрольними точками:

1 етап – створення загальної картини;

1 контрольна точка – затвердження документа загального уявлення;

2 етап – планування;

2 контрольна точка – затвердження проєктних планів;

3 етап – розробка;

3 контрольна точка – остаточне затвердження області дій проєкта;

4 етап – стабілізація (тестування та підготовка до розгортання);

4 контрольна точка – підтвердження готовності проєкту до випуску;
5 етап – розгортання;
5 контрольна точка – рішення розгорнуто.

Використовується багатий практичний досвід Microsoft. Включає опис принципів управління людьми і робочими процесами.

MSF використовує два різновиди моделей:

- модель проєктної групи;
- модель процесів.

MSF вміщує 3 дисципліни:

- дисципліна управління проєктами;
- дисципліна управління ризиками;
- дисципліна управління підготовкою.

Microsoft не використовує в «чистому варіанті» саму методику MSF в своїх IT-проєктах.

Аналіз існуючих моделей та методів

В останні роки успішність проєкту займає все більшу значимість як для творців, так і для замовників проєкту. Практика показує, що немає можливості спланувати та реалізувати довгостроковий проєкт без внесення зміни протягом усього процесу. Тому виникають нові методології, деякі з них більше підходять для невеликих колектив розробників та дрібних замовників (ітераційні методи) – ризики не великі, немає обмежень на час розробки, нечітко визначені цілі розробки (спіральна модель та XP, можливо – Scrum). На практиці більшість ітераційних

методів конкурують один з одним, але деякі (RUP, MSF) дозволяють все ж таки виконувати розробку великих проектів з певними цілями, часом розробки та бюджетом, що стосується великих колективів розробників, оскільки заздалегідь визначено обсяг робіт і, відповідно, забезпечено зайнятість.

Відповідно перші, ітераційні методи розробки отримали назву «легковагових» (*lightweight*), а другі, що вимагають повного визначення всіх характеристик продукту на початковому етапі – «великовагові» (*heavyweight*).

3. УПРАВЛІННЯ ЯКІСТЮ

Для будь-якого підприємства основним джерелом існування є успішна реалізація якісного продукту.

Говорячи про якість продукту, завжди мається на увазі споживач, оскільки саме споживач визначає прийнятні властивості товару.

В умовах ринкової економіки якість продукту – це завдання номер один, оскільки конкурентність фірми визначається:

1. Рівнем ціни продукції.
2. Рівнем якості продукції.

Причому, якість продукції поступово виходить на перше місце.

Параметрів, які визначають якість товару, безліч, тому в менеджменті виникла необхідність розвитку такої течії, як управління якістю.

> ***Управління якістю*** *– це діяльність з управління всіма етапами життєвого циклу продукції, а також взаємодією із зовнішнім середовищем.*

На даний момент у всьому світі розроблено безліч (кілька сотень) систем контролю якості продукції. Загальне завдання цих систем – відповідність продукції вимогам споживачів.

Найбільш відомим і широко використовується стандартом для організації процесів контролю якості є серія стандартів **ISO 9000**.

Для розробки програмного забезпечення використовується керівництво ISO 9000-3, який містить опис процесу розробки для досягнення потрібного рівня якості.

Наразі рекомендується використання стандарту версії 2014 року. Головна увага тепер приділяється управлінню процесом. Поки що стандарт не містить частини для розробників ПЗ.

Впровадження стандартів ISO 9000 створює базу для незалежної сертифікації продукції, яка орієнтована на підтвердження рівня якості продукції, що визначає її конкурентні можливості.

Недолік стандарту ISO 9000 – немає можливості адекватно оцінити рівень якості процесу розробки ПЗ для запропонованої моделі оцінки якості.

Альтернативна модель якості (США): **CMM – SEI (Software Engineering Institute)**. Ця модель якості розроблена в інституті інженерії програмного забезпечення для використання державними, зокрема військовими організаціями при розміщенні замовлень на розробку програмного забезпечення.

Зараз модель CMM – SEI застосовується для аналізу та сертифікації процесів розробки програмного забезпечення фірмами, що виробляють найбільш складні розробки в програмуванні.

Історія розвитку систем якості

В історії розвитку систем якості можна виділити наступні п'ять етапів:

- **1905 р.: Система Тейлора** – система вимог до якості виробів у вигляді допусків (граничне мінімальне та

максимальне значення). Містить шаблони, налаштовані на верхню та нижню межі допусків. Це система управління якістю кожного окремого виробу;

- **1924 р.:** Фірмою Bell Telephone Laboratories закладено **основи статистичного управління якістю**. Пізніше статистичні методи контролю якості набули широкого загалу. Для Японії статистичні методи контролю стали основою економічної революції. При статистичних методах контролю якості, замість перевірки та виявлення дефектів, головна увага приділяється запобіганню дефектам шляхом усунення їх причин;
- **1950-ті рр.:** З'явилася ідея **тотального (загального) контролю якості – TQC (Total Quality Control).** Автор – американський вчений **А. Фейгенбаум** (1957 р., стаття «*Комплексне управління якістю*»). Головні завдання TQC:
  1. Передбачуване виправлення можливих невідповідностей продукції ще на стадії конструювання.
  2. Перевірка якості продукції, комплектуючих та матеріалів, що поставляються.
  3. Керування виробництвом.
  4. Застосування служби сервісного обслуговування.
  5. Спостереження за відповідністю пред'явленим вимогам.

На цьому етапі з'явилися документовані системи якості, що встановлюють відповідальність, повноваження та взаємодію в сфері якості керівництва підприємства та спеціалістів служб якості. В Європі з'явилися аудитори (незалежна третя сторона, що виконує реєстрацію та

сертифікацію систем) і велику увагу стали приділяти документуванню систем забезпечення якості. Японія зустріла із захопленням ідеї TQC і зробила кроки для подальшого розвитку.

- **1980-ті рр.: відбувається перехід від тотального контролю якості (TQC) до тотального управління якістю (TQM).**

З'явилася безліч нових міжнародних стандартів системи управління якістю – стандарти ISO 9000 (1987 р.). TQM не просто управляє якістю, TQM передбачає управління і цілями, і вимогами. У TQM забезпечення якості розуміється як система заходів, покликана викликати у замовника програмного продукту впевненість у його якості. Основний принцип TQM – покращенню немає межі. Система управління якістю не вирішує всіх завдань, необхідних для забезпечення конкурентоспроможності, але вона стає все популярніше і сьогодні вона посідає міцне місце у ринковому механізмі.

- **1990-ті рр.: посилюється вплив суспільства на підприємства, а підприємства дедалі більше враховують інтереси суспільства.**

Це призводить до появи стандартів серії ISO 14000, які встановлюють вимоги до систем менеджменту з погляду захисту навколишнього середовища та безпеки продукції.

Стандарти, що з'являються, доповнюються, розширюються і призводять до багатогранного розгляду питання управління якістю продукції. Тепер управління якістю продукції – це не просто контроль якісних параметрів та причин їх відхилень, це управлінська діяльність, що

охоплює життєвий цикл продукції, системно забезпечує стратегічні та оперативні процеси підвищення якості продукції та функціонування самої системи управління якістю.

Що таке якість? Фактори якості

Якість – це міра корисності, доцільності та ефективності праці, що відчувається кожною людиною. Підвищення якості, у свою чергу, призводить до зниження втрат на всіх етапах життєвого циклу продукції (маркетинг > розробка > виробництво > споживання > утилізація). Внаслідок цього знижується собівартість і ціна продукту, а це призводить до підвищення життєвого рівня людей. Зараз існує кілька визначень якості, які загалом сумісні.

Визначення **«якість ПЗ»** в міжнародних стандартах звучить так:

> ***Якість програмного забезпечення*** *– це сукупність характеристик ПЗ, які належать до його здатності задовольняти встановлені та передбачувані потреби.*
>
> [ISO 8402:1994 Quality management and quality assurance]

> ***Якість програмного забезпечення*** *– це ступінь, в якому ПЗ має необхідну комбінацію властивостей.*
>
> [1061–1998 IEEE Standard for Software Quality Metrics Methodology]

Згідно з *wikipedia.org*:

> *«**Фактор якості ПЗ** – це нефункціональна вимога до програми, яка зазвичай не описується в договорі із замовником, проте є бажаною вимогою за-для підвищення якості програми».*
>
> (wikipedia.org)

У таблиці нижче наведено різні **фактори якості**:

Зрозумілість	Призначення ПЗ має бути зрозумілим – програмний інтерфейс та документація
Повнотаww	Програма має бути реалізована повністю
Стислість	Відсутність дублюючої інформації. Відсутність дублюючого коду (заміна на виклик процедур). Відсутність дублювання документації
Портованість	Легкість перенесення ПЗ у нове оточення
Узгодженість	У всій програмі та документації повинні використовуватися однакові формати та позначення
Супроводжуваність	Вимога до документації та наявності резерву зросту при зміні ресурсів (пам'ять, процесор). Наскільки просто змінити програму при змінених вимогах замовника
Тестованість	Програма має виконувати тестові приклади та засоби для вимірювання продуктивності
Зручність використання	Простота та зручність інтерфейсу користувача
Надійність	Відсутність відмов та збоїв у роботі програм, а також простота відновлення робочого стану
Структурованість	Чітко виділені частини програми та зв'язки між ними
Ефективність	Раціональне використання ресурсів
Безпека	Зберігання та обмеження доступу до конфіденційної інформації, фіксація транзакцій

Управління та забезпечення якості – це поняття, яке охоплює всі етапи розробки, випуску та експлуатації ПЗ.

4. ДОКУМЕНТУВАННЯ

Документація на програмне забезпечення – це документи, в яких описані правила встановлення та експлуатації ПЗ.

Документування ПЗ є настільки важливим етапом його розробки, що найчастіше успіх поширення та експлуатації програмного продукту великою мірою залежить саме від якості його документації. Чим складніший і заплутаніший проєкт, тим вище роль супровідних документів. Слід зазначити, що типи документів бувають різні залежно від характеру інформації, яку вони містять.

Можна відзначити п'ять цілей розробки документів:

- документи встановлюють зв'язки між усіма залученими до процесу розробки;
- документи описують обов'язки групи розробки;
- документи дозволяють керівникам оцінювати перебіг розробки;
- документи утворюють основу документації супроводу програмного забезпечення;
- документи описують історію розробки програмного забезпечення.

Якість програмного забезпечення, поряд з іншими факторами, визначається повнотою та якістю пакету документів, що супроводжують ПЗ. В пакеті документів, що супроводжують ПЗ, входять документи, що містять відомості, необхідні для розробки, виготовлення, супроводу та експлуатації програми.

Існує чотири основні типи документації на ПЗ:

- **архітектурна** або **проєктна** – загальний опис програмного забезпечення, робочого середовища та основних принципів створення ПЗ;
- **технічна** – детальна документація на програмний код, застосовані алгоритми, міжмодульні інтерфейси, API;
- **користувацька** – посібники для кінцевих користувачів, адміністраторів системи, монтажників та ін.;
- **маркетингова** – принципи реалізації, бонусні та партнерські додатки тощо.

Будь-яка компанія висуває унікальні вимоги до складу та змісту документів для своєї продукції. Більшість цих вимог ґрунтується на положеннях та рекомендаціях національних стандартів та/або стандартах, поширених у світовій практиці.

Принципи управління документуванням програмного забезпечення однакові для будь-якого обсягу проєкту. Хоча для невеликих проєктів значну частину положень можна не застосовувати, але принципи залишаються такими ж.

Більшість проєктів розробляється на основі 2-3-х документів:

- технічне завдання;
- інструкція користувача;
- інструкція адміністратора.

На вимогу замовника розробник повинен надати повний перелік документів, що відповідає державним або міжнародним стандартам або вимогам контракту.

Документація може розроблятися як для всього програмного комплексу, так і для окремих його складників.

Назвемо необхідні та достатні документи, щоб програмна система була створена з необхідним рівнем якості. При цьому слід зазначити, що:

- **необхідність документа** – документ, без якого неможливо створити якісний продукт;
- **повнота документа** – якщо роботи виконуються за документом, гарантовано вийде продукт.

Процес розробки програмного продукту схожий на інші інженерні завдання. Інститут інженерів з електротехніки та радіоелектроніки (IEEE, www.ieee.org) розробив стандарти документації процесу розробки програмного забезпечення.

Склад документації проєкту відповідно до цих стандартів за групами документів у зазначеній послідовності:

1. **План експертизи програмного забезпечення (SVVP – Software Verification and Validation Plan):** документ описує, яким чином і в якій послідовності мають перевірятися стадії проєкту і сам продукт на відповідність вимогам.

Простіше кажучи, цей документ застерігає те, як замовник і розробник переконаються, що вони зробили те, що хотіли зробити і що вони справді це зробили. Він не визначає, що і як вони робитимуть.

2. **План контролю якості програмного забезпечення (SQAP – Software Quality Assurance Plan):** яким чином проєкт має досягти відповідності встановленому рівню якості.

Тобто розробник в окремому документі описує, що він намагатиметься, щоб у програмному продукті не було помилок. А якщо, незважаючи на всі «організаційні заходи», вони таки виникли, то замовник знатиме, що розробник дуже старався, щоб їх не було.

3. **План управління конфігураціями програмного забезпечення (SCMP – Software Configuration Management Plan):** містить опис, де і яким чином розробник зберігатиме версії документації, програмного коду і яким чином визначатиме, як код пов'язаний з документацією.

Наприклад, якщо програмісту необхідно внести зміни до програмного модуля, йому буває важко знайти опис того, що саме повинен робити цей модуль, що він робив раніше і так далі серед величезної купи різних версій документів. Для допомоги у пошуках потрібен ще один документ, який описує, що саме він повинен робити, щоб знайти потрібну інформацію.

4. **План управління програмним (SPMP – Software Project Management Plan):** описує, яким чином розробник буде керувати проєктом створення програмного продукту, щоб довести розробку до якісного результату. Він містить управління ризиками, календарний план, кадрові питання і т. д.

5. **Специфікація вимог до програмного забезпечення (SRS – Software Requirements Specification):** найважливіший документ, який містить дві частини – вимоги замовника та візуальну структуру реалізації – те, що може бути виконано.

Тому, замовнику надають дві частини для звірки: чи правильно описані усі його вимоги в обох частинах документа.

Після того як замовник підписав цей документ (вирок), вже він буде винен у тому, якщо щось упустив або передумав; у тому, що розробник помилився під час запису вимог або продукт не відповідає бажанням замовника.

6. **Проєктна документація програмного забезпечення (SDD – Software Design Document):** це архітектура ПЗ та деталі проєктування програми. Не надається замовнику. Найчастіше є секретом підприємства.
7. **Документація з тестування програмного забезпечення (STD – Software Test Documentation):** містить опис того, як, ким і коли має проводитися тестування.

Цей підхід до розробки документації вважався єдиним вірним у 80-90-х роках XX сторіччя.

Вже наприкінці 90-х розробники прийшли до розуміння того, що в програмне забезпечення постійно вносяться зміни через зміну вимог і немає необхідності намагатися створити найдовговічнішу програму.

В результаті виник підхід, названий екстремальним програмуванням (***XP – eXtreme Programming***).

Екстремальне програмування дозволяє замовнику:

1. Не визначати всіх вимог заздалегідь.
2. Вносити зміни в процесі розробки.
3. Керувати необхідністю подальшої розробки ПЗ.
4. Не сплачувати непотрібну документацію.

Екстремальне програмування пред'являє досить жорсткі вимоги програмістам і створюваного ним програмного коду, тому що саме код є основною документацією проєкту і його достатньо для успішного супроводу про-

дукту. Екстремальне програмування вимагає кількох складових для документування коду:

- вибір зрозумілих імен функцій, змінних та класів;
- модульні, інтуїтивно зрозумілі тестові випадки;
- розгорнуті коментарі при описах класів та їх складових.

Стандартизація документації

Основна функція технічної документації – це зберігання та отримання всіляких відомостей технічного характеру. Тому технічна документація має бути якомога більш зрозумілою, інформативною, зручною тощо. Мабуть, усі читають саме таку документацію? Написати чітку та зрозумілу документацію дуже складно, але існують стандарти та методики, які спрощують процес.

Технічна документація має ще одну важливу функцію – нормативну: вона фіксує **взаємні зобов'язання учасників розробки**. Після затвердження технічної документації, зміна вмісту в односторонньому порядку юридично неможлива ані замовником, ані розробником.

Приклад 1

Якщо у затвердженому технічному завданні (ТЗ) на автоматизовану систему сказано, що відвідувач сайту має бачити гасло «Слава праці», замовник не має права раптом вимагати ще й демонстрації рекламного ролика на цю тему. Цілком можливо, що показувати ролик – непогана ідея, але замовнику слід було висловити її раніше, коли розробник погоджував із ним технічне завдання. З іншого боку, якщо замість затвердженого тексту система

видаватиме рекламу нічного клубу, у замовника з'являться всі підстави вимагати від розробника внесення до системи необхідних змін, і не платити йому грошей, доки система не буде приведена у відповідність до технічного завдання.

Приклад 2

Якщо у затвердженому технічному проєкті зафіксовано, що для реалізації автоматизованої системи використовується скрипт на PHP, розробник не має права приголомшити замовника пропозицією замість безкоштовного інтерпретатора цієї мови придбати Lotos Domino. Але, з іншого боку, і замовник не може вимагати від розробника замість нормального сервера використовувати комп'ютер «БК-0010», що завалявся в коморі з минулих часів.

Документація з експлуатації є частиною продукту, оскільки без неї неможливо або дуже складно застосовувати ПЗ. Стандартизація документація з експлуатації дозволяє не описувати у договорі чи технічному завданні докладні вимоги до неї. Замість цього використовуються посилання відповідні стандарти або їх частини.

Тому розробка комплекту документації має виконуватися фахівцем високої кваліфікації, який розбирається як у змісті проєкту, так і психології майбутнього користувача. Бажано володіння літературним стилем тощо.

Урок 1
ВСТУП ДО КЕРУВАННЯ ПРОГРАМНИМИ ПРОЄКТАМИ