**Тема 4. Сучасні методології розроблення програмних систем**

**CASE-засоби та нотації моделювання програмних систем**

На становлення технологій виробничого програмування найбільш помітний вплив мали методологія структурного програмування та об'єктно- орієнтоване програмування. Перша дозволила усвідомити обмеженість здібностей людини, необхідних для створення великих програм. Об'єктно- орієнтоване програмування дало поштовх до розроблення методів декомпозиції, пристосованих для подолання складності проектів, та привело до модернізації основних принципів проектування програм. Незважаючи на ці та інші впливи, стадія комплексної автоматизації технологій програмування стала можливою лише при відповідному рівні розвитку техніки. Важливою обставиною, що дозволила перейти до комплексної автоматизації, стало усвідомлення того, що не можна говорити про промислове програмування без підтримки технологічних функцій на усіх етапах життя програм. Приблизно на початку 90-х рр. ХХ ст. з'явився термін CASE-технологія (Computer Aid Software Engineering - комп'ютерна підтримка розроблення програм), яким стали позначати використання систем, що містять комплексні автоматизовані засоби підтримки розроблення і супроводу ПЗ. Найбільш вдалим виявилося використання CASE-систем у тих спеціальних областях, в яких вже були успіхи і досвід технологічної практичної роботи, а також у тих випадках, коли ця область вже була забезпечена надійною теоретичною базою. На рисунку 7.1 наведено етапи розвитку CASE-засобів за областями застосування.

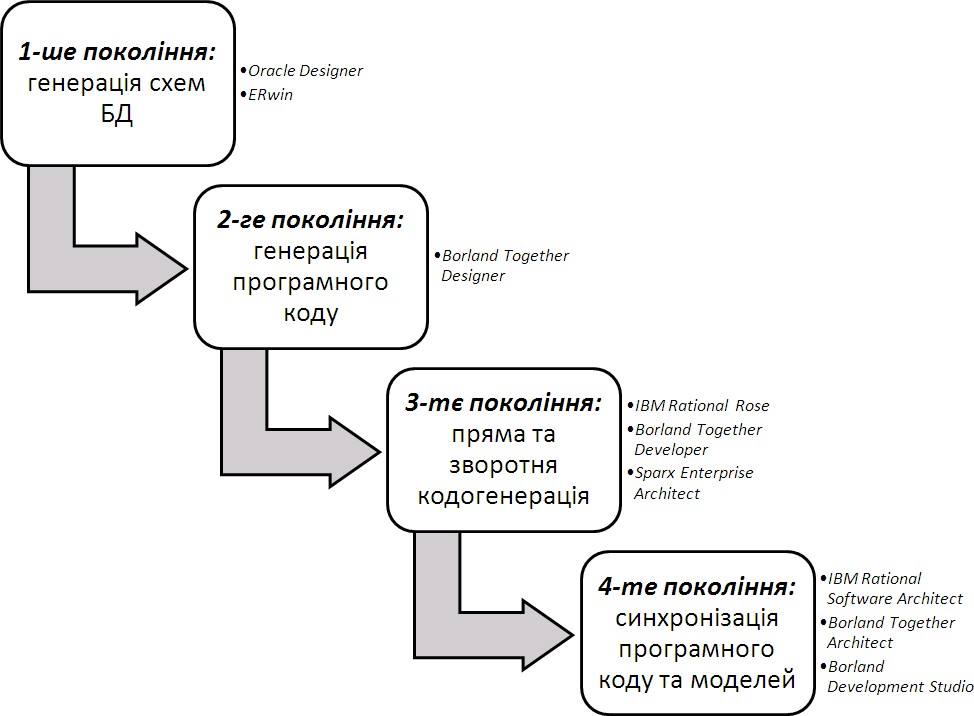


Рисунок 7.1 – Етапи розвитку CASE-засобів за областями застосування

Першими з'явилися CASE-системи розроблення баз даних у розвинених реляційних СУБД (наприклад, Oracle Designer в системі Oracle). Наступними стали засоби генерації програмного коду та налагодження програм. Потреба підтримки не лише процесу програмування, а й етапу аналіз та проектування програмних продуктів привело до створення CASE- систем, які зв’язували моделі ПЗ із програмним кодом – спочатку лише для прямої та зворотної кодогенерації, а потім із підтримкою синхронного зв'язку коду із моделями аналізу та проектування.

Сьогодні універсальні CASE-системи будуються у рамках застосування розвинених, але все ж спеціальних методологій. Безперечний прогрес у даній сфері досягнутий для проектування, орієнтованого на моделювання на етапах аналізу і конструювання (CASE-системи 4-го покоління).

Складність процесу розроблення інформаційних систем викликала появу візуальних засобів моделювання (рис.7.2). У рамках об'єктно- орієнтованого підходу Object management group (OMG) розроблена спеціальна уніфікована мова моделювання UML (Unified Modeling Language), що розглядається як основа проектування в методології ітеративного нарощування можливостей об'єктно-орієнтованих програмних систем. Широкого вжитку для моделювання процесів різних галузей екномічної діяльності набула родина візуальних мов IDEF (Integration Definition), що є стандартом моделювання Міністерства оборони США. Паралельно із нотаціями UML та IDEF розвиваються методологія та нотація для професійного моделювання бізнес-процесів ARIS (ARchitecture of Integrated Information Systems.

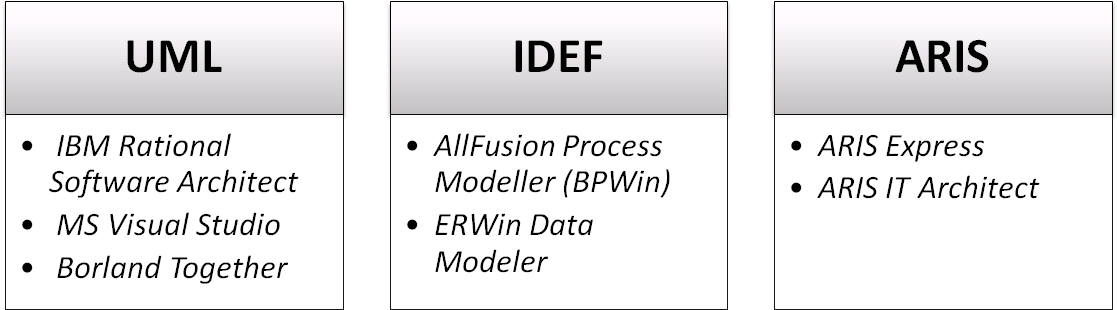


Рисунок 7.2 – Нотації візуального моделювання систем

*Зміна життєвого циклу програмного забезпечення при використанні CASE-технологій*

CASE-технології являють собою сукупність методологій аналізу, проектування, розробки та супроводу складних програмних систем, які засновані як на структурному, так і на об'єктному підходах, що підтримуються комплексом взаємопов'язаних засобів автоматизації. В основі будь якої CASE-технології лежить парадигма методологія / метод / нотація / засіб.

Методологія будується на базі деякого підходу і визначає кроки роботи, їх послідовність, а також правила розподілу та призначення методів.

Метод визначає спосіб досягнення тієї чи іншої мети - виконання кроку роботи.

Нотацією називають систему позначень, використовуваних для опису деякого класу моделей. Нотації бувають графічні (надання моделей у вигляді графів, діаграм, таблиць, схем і т. п.) і текстові (описи моделей на формальних і природних мовах). В CASE-технологіях нотації використовують для опису структури проектованої системи, елементів даних, етапів обробки і т. п. графічного проекту, організації проекту у вигляді ієрархії рівнів абстракції, а також перевірки відповідності компонентів різних рівнів.

Розрізняють:

CASE-засоби аналізу вимог, проектування специфікацій і структури; редагування інтерфейсів (перше покоління CASE-I);

CASE-засоби генерації вихідних текстів та реалізації інтегрованого оточення підтримки повного життєвого циклу розробки програмного забезпечення (друге покоління CASE-II).

CASE-I в основному включають кошти для підтримки графічних моделей, проектування специфікацій, екранних редакторів і словників даних.

CASE-II відрізняється істотно великими можливостями, забезпечуючи: контроль, аналіз і зв'язування системної інформації та інформації з управління процесом проектування, побудова прототипів і моделей системи, тестування, верифікацію і аналіз згенерованих програм.

Автоматизуючи трудомісткі операції, сучасні CASE-засоби суттєво підвищують продуктивність праці програмістів і покращують якість створюваного програмного забезпечення.

Вони:

* забезпечують автоматизований контроль сумісності специфікацій проекту;
* зменшують час створення прототипу системи; прискорюють процес проектування і розробки;
* автоматизують формування проектної документації для всіх етапів життєвого циклу відповідно до сучасних стандартів;
* частково генерують коди програм для різних платформ розробки;
* підтримують технології повторного використання компонентів системи;
* забезпечують можливість відновлення проектної документації за наявними вихідними кодами.

Поява CASE-технологій змінила всі етапи життєвого циклу програмного забезпечення, при цьому найбільші зміни стосуються аналізу і проектування, які припускають суворий і наочний опис розроблюваного програмного забезпечення. Використання CASE-засобів дозволяє істотно знизити трудовитрати на розробку складного програмного забезпечення в основному за рахунок автоматизації процесів документування та контролю. Проте слід мати на увазі, що сучасні CASE-засоби дорогі, а їх використання вимагає *більш високої кваліфікації розробників*. Отже, їх має сенс використовувати в складних проектах, причому, чим складніше розроблюване програмне забезпечення, тим більше виграш від використання CASE-технологій.

На сьогоднішній день практично все промислово вироблене складне програмне забезпечення розробляється з використанням CASE-засобів.

**Жорсткі та гнучкі стратегії в методологіях програмування**

Визначаючи стратегії послідовного і ітеративного розвитку проектів. Контроль діяльності проекту – завдання, що вимагає спеціальних організаційних підходів.

*Загальною метою є перетворення створення програмного продукту в упорядкований процес****,*** у рамках якого розвиток проекту можна зробити більш прогнозованим і ефективним. Для цього зазвичай створюється детальний опис процесу розробки системи, особливе місце в якому займає планування. Інакше кажучи, *створюється метод****,*** *за допомогою якого передбачається конструювати систему****.*** Вдалі методи узагальнюються, в результаті досвід їх застосування перетворюється в методику, або, як зараз прийнято говорити, *методологію****.***

Поряд з усім корисним ці рішення оголошуються обов'язковими, стандартними, їх змушені застосовувати і тоді, коли вихідні передумови втрачені. Щоб слідувати таким методологіям, доводиться виконувати безліч різних приписів, що уповільнює темп основних програмістських робіт. Тому їх називають *важкими****,*** *жорсткими або монументальними****.***

*Жорсткі методології* привабливі для замовників програмних проектів, які завжди можуть перевірити, чи дійсно процес розробки впорядкований і результати відповідають планам. Природно пов'язувати цю можливість з організацією, яка отримує замовлення і забезпечує його менеджмент, оскільки крім найкращих побажань (найчастіше вони-то і порушуються!) в організації можливий адміністративний контроль.

Пропозиції CMM для визначення зрілості організації спираються на те, які процедури управління програмними проектами, відстеження їх розвитку та інші менеджерські методи прийняті як фірмовий стандарт. Методології програмування, що побудовані на базі цих пропозицій, часто розглядають як еталон жорсткості.

На противагу жорстким методологіям останнім часом сформувався компромісний підхід, методології якого об'єднані загальним терміном **agile development.** На російську мову його перекладають як *швидкий розвиток*, *гнучкі методології* і навіть ***"****спритні технології****".*** У новому підході намагаються надати можливість полегшеної організованої роботи колективів програмістів, коли перевантаженість стандартизованого процесу перешкоджає ефективності.

Вони претендують на те, що їх застосування можливе навіть коли не вдається досить точно представити проект при його зародженні, тобто в досить типовій ситуації невизначеності реальної користувальницької потреби.

У цих випадках пропонується залучення замовника до формування завдань та коригуванні припущень протягом всього розвитку проекту. З його допомогою намагаються виявляти найбільш точно і без зайвих бюрократичних процедур актуальні потреби користувачів, що склалися на поточний момент. В результаті з'являється можливість вказати саме ті вимоги, реалізація яких необхідна і допускає максимально короткі терміни випуску релізу.

Всі методології швидкого розвитку орієнтуються на стратегію ітеративного нарощування можливостей системи. Всі вони надають розробникам значно більшу свободу, ніж, приміром, вимоги стандартів CMM. Але не слід думати, що цей підхід повністю скасовує жорсткі методології – *необхідний баланс між свободою швидких методологій і дисципліною****.***

*Для жорстких методологій характерно* прагнення забезпечити розробників рецептами, що не вимагають обговорень: потрібно виконувати відомі рецепти, дотримуватися регламенту, щоб відповідні операційні маршрути приводили до допустимих траєкторій. Іншими словами, *жорсткі методології підтримують* переважно такі діяльності, які можна назвати *імперативними*.

На відміну від традиційних підходів *швидкі методології орієнтуються на те,* що діяльність з виробництва програмного забезпечення по суті своїй є переважно *реативною****,*** тобто такою, в якій від розробників потрібно не тільки розпізнавання ситуацій і застосування в них відомих методів, але і конструювання нових методів дії. А це інший, більш високий рівень знань і умінь.

В даний час широке застосування отримують так звані промислові технології створення програмного продукту. Ці технології були розроблені фірмами, які накопичили великий досвід створення ПЗ. Технології представлені описами принципів, методів, процесів і операцій, що використовуються. Такі технології, як правило, підтримуються набором CASE-засобів, охоплюють всі етапи життєвого циклу продукту і успішно застосовуються для вирішення практичних завдань. Найбільш помітними є такі методології розроблення ПЗ:

***Rational Unified Process (RUP)*** – розробка фірми Rational, довгий час успішно займалася створенням CASE-засобів, що застосовуються на різних етапах життєвого циклу продукту від аналізу до тестування і документування.

*Rational Unified Process (RUP),* що пропонує ітеративну модель розроблення, що включає чотири фази: початок, дослідження, побудову та впровадження. Проходження через чотири основні фази називається циклом розроблення, кожен цикл завершується генерацією працездатної версії системи. У ході супроводу продукт продовжує розвиватися і знову проходить ті самі фази. Розроблення ПЗ на базі RUP базується на створенні та супроводі моделей на базі UML.

***Microsoft Solution Framework (MSF)*** – розробка фірми Microsoft, призначена для вирішення широкого кола завдань. Технологія масштабована, тобто налагоджується на вирішення завдань будь-якої складності колективом будь-якої чисельності.

*Microsoft Solution Framework (MSF)* – методологія, що розроблялася для підвищення керованості процесів розроблення окремого підприємства (Microsoft), а потім стала застосовуватися розробниками, які використовують продукти Microsoft. Методологія, подібна до RUP, також включає чотири фази: аналіз, проектування, розроблення, стабілізацію, є ітераційною, припускає використання об'єктно-орієнтованого моделювання. MSF порівняно з RUP здебільшого орієнтована на розроблення бізнес-додатків.

***Extreme Programming (XP)*** – активно розвивається останнім часом технологія, що призначена для вирішення відносно невеликих завдань, відносно невеликими колективами професійних розробників в умовах жорстко обмеженого часу.

*eXtreme Programming (XP)* – методологія, що приділяє основну увагу ефективній комунікації між замовником і виконавцем упродовж усього проекту з розроблення ІС для відслідковування зміни вимог. В основу підходу покладена командна робота та постійне тестування прототипів.

*Гнучке розроблення ПЗ* ***(Agile)*** – клас методологій розроблення програмного забезпечення на основі ітеративної моделі ЖЦ, в якій вимоги та рішення еволюціонують через співпрацю між самоорганізовуваними командами.

Кожна з цих технологій має свої особливості організації моделі життєвого циклу створення продукту. Розглянемо особливості моделей життєвого циклу трьох найбільш відомих промислових технологій:

**Методологія Rational Unified Process (RUP). Методологія Microsoft Solution Framework (MSF). Методологія eXtreme Programming (XP).**

*Методологія* **Rational Unified Process (RUP)**

Уніфікований процес розроблення ПЗ Rational Unified Process (RUP) виник як відповідь на потребу розробників у процесі, що об'єднує безліч аспектів розроблення програм і відповідає таким вимогам:

* + забезпечує керівництво діяльністю команди;
  + керує завданнями окремого розробника і команди в цілому;
  + показує, які артефакти необхідно розробити;
  + надає критерії відстеження та вимірювання продуктів і функціонування проекту.

Уніфікований процес розроблення ПЗ передбачає розподіл ЖЦ програмного забезпечення на фази (рис. 8.1).

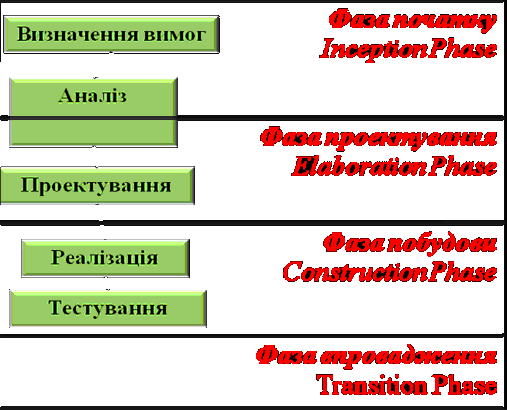


Рисунок 8.1 – Робочі процеси ЖЦ в уніфікованому процесі розроблення

ПЗ

Специфіка уніфікованого процесу полягає у трьох словосполученнях –керований варіантами використання, архітектурно-орієнтований, ітеративний та інкрементний.

Модель життєвого циклу RUP є досить складною, детально відпрацьованою ітеративно-інкрементною моделлю з елементами каскадної моделі. У моделі RUP виділяються 4 основні фази, 9 видів діяльності (процесів). Крім того, в моделі описується ряд *практик,* які слід застосовувати або керуватися для успішного виконання проекту. RUP орієнтований на поетапне моделювання продукту, що створюється, за допомогою мови UML.

*Основними фазами* ***RUP*** *є:*

* + - Фаза початку проекту **(Inception).** Визначаються основні цілі проекту, бюджет проекту, основні засоби його виконання - технології, інструменти, ключовий персонал, складаються попередні плани проекту. Основна мета цієї фази - досягти компромісу між усіма зацікавленими особами щодо завдань проекту.
    - Фаза опрацювання **(Elaboration).** Основна мета цієї фази - на базі основних, найбільш істотних вимог, розробити стабільну базову архітектуру продукту, яка дозволяє вирішувати поставлені перед системою завдання і надалі використовується як основа розробки системи.
    - Фаза побудови **(Construction).** Основна мета цієї фази - детальне прояснення вимог і розробка системи, що задовольняє їм, на основі спроектованої раніше архітектури.
    - Фаза передачі **(Transition).** Мета фази - зробити систему повністю доступною кінцевим користувачам. Тут відбувається остаточне розгортання системи в її робочому середовищі, підгонка дрібних деталей під потреби користувачів. В рамках кожної фази можливе проведення декількох ітерацій, кількість яких визначається складністю виконуваного проекту.

*Діяльності* ***(****основні процеси****) RUP*** діляться на п'ять робочих і чотири підтримуючі.

*До робочих діяльностей відносяться****:***

* + - *Моделювання предметної області* (бізнес-моделювання, Business Modeling). Цілі цієї діяльності - зрозуміти бізнес-контекст, в якому повинна буде працювати система (і переконатися, що всі зацікавлені особи розуміють його однаково), зрозуміти можливі проблеми, оцінити можливі їх рішення та їх наслідки для бізнесу організації, в якій працюватиме система.
    - *Визначення вимог* ***(Requirements)****.* Цілі - зрозуміти, що повинна робити система, визначити межі системи і основу для планування проекту та оцінок ресурсозатрат в ньому.
    - *Аналіз та проектування* ***(Analysis and Design).*** Вироблення архітектури системи на основі ключових вимог, створення проектної моделі, представленої у вигляді діаграм UML, що описують продукт з різних точок зору.
    - *Реалізація* ***(Implementation).*** Розробка вихідного коду, компонент системи, тестування та інтегрування компонент.
    - *Тестування* ***(Test).*** Загальна оцінка дефектів продукту, його якість в цілому; оцінка ступеня відповідності вихідним вимогам.

*Підтримуючими діяльностями є****:***

* + - *Розгортання* ***(Deployment).*** Цілі – розгорнути систему в її робочому оточенні і оцінити її працездатність.
    - *Управління конфігураціями і змінами* ***(Configuration and Change Management)****.* Визначення елементів, що підлягають зберіганню та правил побудови з них узгоджених конфігурацій, підтримку цілісності поточного стану системи, перевірка узгодженості внесених змін.
    - *Управління проектом* ***(Project Management).*** Включає планування, управління персоналом, забезпечення зв'язків з іншими зацікавленими особами, управління ризиками, відстеження поточного стану проекту.
    - *Управління середовищем проекту* ***(Environment)****.* Налаштування процесу під конкретний проект, вибір і зміна технологій та інструментів, що використовуються в проекті.

*Методологія* **Microsoft Solution Framework (MSF)**

Одна з особливостей технології MSF полягає в тому, що вона орієнтована не просто на створення програмного продукту, що задовольняє перерахованим вимогам, а на пошук рішення проблем, що стоять перед замовником. Різниця полягає в тому, що перелічені замовником вимоги є проявами деяких більш глибоких проблем і неточність, неповнота, зміна вимог у процесі розробки - наслідок недостатнього розуміння проблем. Тому, в технології MSF велика увага приділяється аналізу проблем замовника і розробці варіантів системи для пошуку рішення цих проблем.

Модель життєвого циклу MSF є деяким гібридом каскадної і спіральної моделей, поєднуючи простоту управління каскадної моделі з гнучкістю спіральної.

*Модель життєвого циклу MSF орієнтована на "віхи" (milestones) – ключові точки проекту, що характеризують досягнення якого-небудь істотного результатую.*

Цей результат може бути оцінений і проаналізований, що має на увазі відповідь на питання: "А чи досягли ми цілей, поставлених на цьому кроці?».

*В основу MSF покладено вісім базових принципів (foundational principles):*

− *Сприяння відкритої комунікації* ***(Foster open communications).*** Модель процесу MSF запроваджує вільний інформаційний потік сред членів команди та зацікавлених сторін (key stakeholders) для забезпечення однакового розуміння завдань. Документування ходуи проекту та доступ до цих даних членів команди, зацікавлених сторін та замовників.

− *Робота у напрямку спільного бачення проекту* ***(Work toward a shared vision)***. Модель процесу MSF запроваджує фазу формування концепції (Envisioning Phase) та окремий ключовий момент затвердження бачення (milestone Vision/Scope Approved) для формування спільного бачення проекту. Бачення включає детальне розуміння цілей та завдань для досягнення рішення поставленої проблеми. Спільне бачення виявляє припущення команди та замовників, потрібні для отримання рішення.

− *Надання прав і можливостей членам команди* ***(Empower team members)***. Розширення прав і можливостей членів команди для прийняття ними відповідальності за виконану роботу. Таке збільшення відповідальності може бути затверджене у графіках, де фіксується дата закінчення робіт, що також може бути засобом виявлення можливих затримок проекту.

− *Визначення індивідуальної та спільної відповідальності* ***(Establish clear accountability and shared responsibility)***. Модель командної групи MSF ґрунтується на принципі важливості роботи кожного для отримання якісного рішення проблеми. Усі члени групи розділяють відповідальність за проект.

− *Зосередження на бізнес****-****цілях* ***(Focus on delivering business value).*** Рішення повинне приносити користь організації у вигляді додавання вартості бізнесу. Це додавання досягається тільки після повного розгортання рішення у виробничому середовищі.

− *Бути гнучкими****,*** *очікувати на зміни* ***(Stay agile, expect change).*** MSF припускає, що у виробничому середовищі на рішення постійно впливають зміни. Команда повинна бути обізнаною та готовою до керування змінами вимог.

− *Інвестування у якість* ***(Invest in quality)***. У MSF кожен член команди відповідальний за якість вирішення завдання.Для підтримки якості протягом проекту формується команда тестувальників. Це гарантує, що рішення відповідає рівню якості, визначеному зацікавленими сторонами.

− *Навчання за досвідом* ***(Learn from all experiences).*** MSF вимагає використання досвіду, отриманого у попередніх проектах. Це дозволяє знайти найкращі методики розроблення.

***MSF*** *складається із двох моделей та трьох дисциплін****:***

− моделі командної групи;

− моделі процесу;

− дисципліни управління проектами;

− дисципліни управління ризиками;

− дисципліни управління підготовкою.

*Модель командної групи* ***(MSF Team Model)*** описує, як організувати колективи і якими принципами керуватися для досягнення максимального успіху в розробленні програм. Різні колективи можуть по-своєму застосовувати на практиці різні елементи цієї моделі – все залежить від масштабу проекту, розміру колективу і кваліфікації його учасників та моделі процесу розроблення.

Формування колективу є складним завданням, що повинне виконуватися психологами та враховувати такі *основні положення*:

− не повинне бути команди з одних лідерів;

− не повинне бути команди з одних виконавців;

− у випадку невдачі команда розформовується;

− система штрафів (якщо проект провалюється – карають усіх).

Модель командної групи визначає тільки ролі, кожна з яких може виконуватися кількома людьми (рис. 8.2). Цікаво, що в моделі командної групи не передбачено єдиноначальності – усі ролі важливі, усі ролі рівноправні, тому MSF називають моделлю рівних (team of peers).

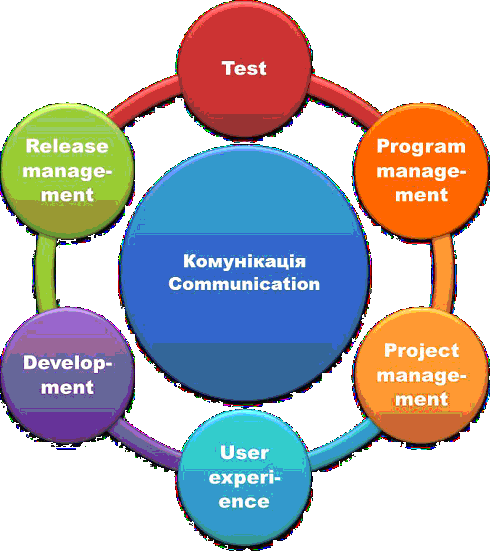


Рисунок 8.2 – Модель командної групи (MSF Team Model)

*Program management* – керування програмою. Виконавець цієї ролі відповідає за організацію (але не керує): здійснює ведення графіка робіт, ранкові 15-хвилинні наради, забезпечує відповідність стандартам і специфікаціям, фіксацію порушень, написання технічної документації.

*Product management* – керування продуктом. Виконавці цієї ролі відповідають за спілкування із замовником, написання специфікації, роз'яснення завдань розроблювачам.

*Development* – найбільш традиційна роль – розроблення і початкове тестування продукту.

*User expirience* – підвищення ефективності роботи користувачів, написання користувальницької документації.

*Release management* – розгортання релізу продукту, супровід і його технічна підтримка.

*Test* – визначення відповідності показників якості релізу встановленим значенням. Виявлення й усунення недоробок, виправлення помилок, інші функції QA.

У MSF затверджується, що таку модель можна масштабувати, розбиваючи систему за функціями.

Модель процесу (MSF Process Model) визначає, коли і які роботи повинні бути виконані.

*Основні принципи і практичні прийоми****,*** *на яких ґрунтується модель****:***

− ітеративний підхід (послідовний випуск версій);

− підготовка чіткої документації;

− урахування невизначеності майбутнього;

− облік компромісів;

− керування ризиками;

− підтримка відповідального відношення колективу до строків випуску продукту;

− розбивка великих проектів на більш дрібні керовані частини;

− щоденне складання проекту;

− постійний аналіз ходу робіт.

***Process model*** *має три основні особливості****:***

− розбивка всього процесу на фази;

− використання опорних точок (milestones);

− ітеративність.

Увесь процес розбивається на п’ять взаємозалежних фаз (рис. 8.3). Перш ніж переходити до наступної фази, на попередній повинні бути отримані певні результати (досягнуті головні опорні точки). Фактично процес ітеративний і відповідає спіральній моделі ЖЦ ПЗ. У моделі передбачається наявність основних віх (завершення головних фаз моделі) і проміжних, що відображають внутрішні етапи головних фаз.



Рисунок 8.3 – Модель процесу (MSF Process Model)

* + *Створення загальної картини додатку* ***(Envisioning).*** На цьому етапі вирішуються такі основні завдання:

− оцінка існуючої ситуації;

− визначення складу команди;

− структури проекту, бізнес-цілей, вимог і профілів користувачів;

− розробка концепції рішення і оцінка ризику.

Встановлюються дві проміжні віхи: "Організовано кістяк команди" і "Створена загальна картина рішення".

* + *Планування* ***(Panning).*** Включає планування і проектування продукту. На основі аналізу вимог розробляється проект і основні архітектурні рішення, функціональні специфікації системи, плани і календарні графіки, середовища розробки, тестування та пілотної експлуатації.

Етап складається з трьох стадій: концептуальне, логічне і фізичне проектування.

На стадії концептуального проектування задача розглядається з точки зору користувача і бізнес-вимог і закінчується визначенням набору сценаріїв використання системи.

*При логічному проектуванні* задача розглядається з точки зору проектної команди, рішення представляється у вигляді набору сервісів.

І вже на стадії фізичного проектування задача розглядається з точки зору програмістів, уточнюються технології і інтерфейси, що використовуються.

* + *Розробка* ***(Developing).*** Створюється варіант вирішення проблеми, у вигляді коду і документації чергового прототипу, включаючи специфікації і сценарії тестування. Основна віха етапу – "Остаточне затвердження області дії проекту".

Продукт готовий до зовнішнього тестування і стабілізації. Крім того, замовники, користувачі, співробітники служби підтримки та супроводу, а також ключові учасники проекту можуть попередньо оцінити продукт і вказати всі недоліки, які потрібно усунути до його поставки.

* + *Стабілізація* ***(Stabilizing).*** Підготовка до випуску остаточної версії продукту, доведення його до заданого рівня якості. Тут виконується комплекс робіт з тестування (виявлення та усунення дефектів), перевіряється сценарій розгортання продукту. Коли рішення стає досить стійким, проводиться його пілотна експлуатація в тестовому середовищі із залученням користувачів і застосуванням реальних сценаріїв роботи.
  + *Розгортання* ***(Deploying).*** Виконується установка рішення і необхідних компонентів оточення, проводиться його стабілізація в промислових умовах і передача проекту групі супроводу. Крім того, аналізується проект в цілому на предмет рівня задоволеності замовника.

Важливу роль відіграють *опорні точки* ***(milestones),*** у яких аналізується стан робіт і виробляється їхня синхронізація. У цих точках додаток або його специфікації не заморожуються. Опорні точки дозволяють проаналізувати стан проекту і внести необхідні корективи, наприклад, переналагоджуватися під вимоги, що змінилися, замовника або відреагувати на ризики, можливі в ході подальшої роботи. Для кожної опорної точки визначається, які результати повинні бути отримані до цього моменту.

Кожна фаза процесу розроблення завершується головною опорною точкою (major milestone) – моментом, коли всі члени колективу синхронізують отримані результати. Призначення таких точок у тому, що вони дозволяють оцінити життєздатність проекту. Їхні результати видимі не тільки колективу розроблювачів, але й замовникові. Після аналізу результатів колектив розробників і замовник спільно вирішують, чи можна переходити на наступну фазу. Таким чином, головні опорні точки - це критерії переходу з однієї фази проекту на іншу.

Усередині кожної фази визначаються проміжні опорні точки (interium milestones). Вони, як і головні, слугують для аналізу й синхронізації досягнутого, а не для заморожування проекту. Але на відміну від головних опорних точок проміжні видні тільки членам колективу розробників.

Ітеративність процесу MSF полягає в його багаторазовому повторенні упродовж усього циклу розроблення та існування продукту. На кожній успішній ітерації у продукт включаються тільки ті нові інструменти та функції, які задовольняють постійно змінювані вимоги бізнесу.

*Методологія* **eXtreme Programming (XP)**

Екстремальне програмування є прикладом так званого методу «живої розробки» *(Agile Development Method).*

*Екстремальне програмування* ***(eXtreme Programming, XP)*** – спрощена методологія організації виробництва для невеликих і середніх за розміром команд розробників, які займаються розробленням програмного продукту в умовах незрозумілих або швидко змінних вимог.

Програмування відповідно до методик ХР доводить використання загальноприйнятих принципів програмування до екстремальних рівнів:

* перегляд коду виконується постійно (з урахуванням того, що програмування ведеться парами);
* кожен учасник проекту тестує код програми постійно (тестування модулів), навіть замовники проводять функціональне тестування;
* проектування є складовою частиною повсякденної роботи кожного розробника (перероблення коду);
* розроблення виконується з урахуванням вимоги збереження в системі найбільш простого дизайну, що забезпечує поточний необхідний рівень функціональності (простіші речі надійніші в роботі);
* увага до архітектури системи на кожному етапі проекту;
* інтеграційне тестування виконується після кожної найменшої зміни у системі (триваюча інтеграція);
* ітерації невеликі – тривають години, кілька днів (постійне планування).

*Для формування стилю розроблення, що дозволить досягти потрібної якості рішення, в* ***XP*** *пронується керуватись чотирма цінностями*:

* + *комунікацією (communication)* – дисципліна XP, спрямована на забезпечення безперервної комунікації усіх учасників проекту. Під час тестування, програмування в парах та попереднього оцінювання робіт замовники, розробники та менеджери змушені постійно спілкуватись;
  + *простотою (simplicity)* – неможливо заздалегідь точно визначити, як буде розвиватися проект, тому вирішувати необхідно лише завдання, що виникають сьогодні, у найбільш простий спосіб;
  + *зворотним зв'язком (feedback)* – спосіб отримати точні та конкретні дані про стан проекту. Постійне виконання тестів для перевірки усіх змін у системі забезпечує програміста звороним зв’язком про якість роботи. Кожен новий опис замовником вимог до системи одразу ж оцінюється розробниками і забезпечує замовника зворотним зв’язком із інформацію про якість опису. Менеджер, який контролює строки виконання робіт, забезпечує усіх учісників проекту інформацією про виконання планових термнів розроблення. В XP найбільш важливі функції системи реалізуються в першу чергу у реально працюючий продукт, тому замовник досить швидко може оцінити хід виконання та якість розроблення, а також відповідність замовленню;
  + *хоробрість (courage)* – для того, щоб розроблення не втратила своєї актуальності, в XP рішення повинні прийматися з максимальною швидкістю, для чого потрібна певна сміливість.

Для визначення методів вирішення проблем розроблення ПЗ на основі обраних цінностей потрібно керуватися такими *фундаментальними принципами оцінки методів вирішення поставленого завдання****:***

* + *швидкий зворотний зв'язок* – інформація про стан системи повинна якомога швидше надходити до зацікавлених сторін проекту. У рамках дисципліни XP ці дані повинні із максимальною швидкістю надходити, інтерпретуватися та на основі їх аналізу швидко виконуватися модифікації системи;
  + *прийнятна простота* – кожна проблема повинна вирішуватись у найбільш простий спосіб. У рамках XP значні зусилля (тестування, переробка коду, комунікації) покладаються для вирішення завдань сьогодення таким чином, щоб завтра внесення змін не потребувало великих зусиль;
  + *поступова зміна* – кардинальні зміни часто приводять до провалу проекту, щоб заплановані модифікації мали успіх, їх потрібно реалізовувати як серію невеликих змін, після кожної з яких перевіряється працездатність системи;
  + *прийнятна зміна* – найвигідніша стратегія – та, що дозволяє вирішити найбільш важливу проблему і залишає максимальну свободу дій;
  + *якісна робота –* кожен учасник проекту повинен прагнути максимально якісно виконувати свої завдання і сприяти якісній роботі інших.

Кожен принцип втілює заявлені цінності. З-поміж альтернативних методів, які втілюють принципи, в XP пропонується обирати метод рішення, який відповідає виконанню більшості принципів.

*Модель життєвого циклу* ***XP*** є ітераційно-інкрементною моделлю швидкого створення (і модифікації) прототипів продукту, що задовольняють черговій вимозі (user story).

*Основними фазами моделі можна вважати****:***

* + ***«****Вкидання****»*** *архітектури* - початковий етап проекту, на якому створюється бачення продукту, приймаються основні рішення з архітектури та технологій, що застосовуються. Результатом початкового етапу є метафора (metaphor) системи, яка в досить простому і зрозумілому команді вигляді повинна описувати основний механізм роботи системи.
  + *Історії використання* ***(User Story)*** - етап збору вимог, що записується на спеціальних картках у вигляді сценаріїв виконання окремих функцій. Історії використання є вимогами для планування чергової версії і одночасної розробки приймальних тестів (Acceptance tests) для її перевірки.
  + *Планування версії* ***(****релізу****).*** Проводиться на зборах за участю замовника шляхом вибору User Stories, які увійдуть в наступну версію. Одночасно приймаються рішення, пов'язані з реалізацією версії. Мета планування - отримання оцінок того, що і як можна зробити за 1-3 тижні створення наступної версії продукту.
  + *Розробка* проводиться відповідно до плану і включає тільки ті функції, які були відібрані на етапі планування.
  + *Тестування* проводиться за участю замовника, який бере участь у складанні тестів.
  + *Випуск релізу* - розроблена версія передається замовнику для використання або бета-тестування.

По завершенню циклу робиться перехід на наступну ітерацію розробки. Особливості моделі життєвого циклу XP прояснюють наступні принципи цього методу. Насамперед, це принципи «живої» розробки ПЗ, що зафіксовані в маніфесті «живої» розробки:

* + люди та їх спілкування більш важливі, ніж процеси та інструменти;
  + працююча програма більш важлива, ніж вичерпна документація;
  + співпраця із замовником більш важлива, ніж обговорення деталей контракту;
  + відпрацювання змін більш важливе, ніж дотримання планів.

*Крім того, в XP є кілька правил* ***(****технік****),*** *що характеризують особливості моделі його життєвого циклу:*

* + *Живе планування* ***(planning game) –*** якнайшвидше визначити обсяг робіт, який потрібно зробити до наступної версії ПЗ. Рішення приймається на основі, в першу чергу, бізнес-пріоритетів замовника і, по-друге, технічних оцінок. Плани змінюються, як тільки вони починають розходитися з дійсністю або побажаннями замовника.
  + *Часта зміна версій* ***(small releases) –*** перша працююча версія повинна з'явитися якомога швидше, і тут же повинна почати використовуватися. Наступні версії готуються через досить короткі проміжки часу.
  + *Прості проектні рішення* ***(simple design)*** – у кожен момент часу система повинна бути сконструйована так просто, наскільки це можливо. Нові функції додаються тільки після ясного прохання про це. Вся зайва складність вилучається, як тільки виявляється.
  + *Розробка на основі тестування* ***(test-driven development)*** - спочатку пишуться тести, потім реалізуються модулі так, щоб тести спрацьовували. Замовники заздалегідь пишуть тести, що демонструють основні можливості системи, щоб можна було побачити, що система дійсно запрацювала.
  + *Постійна переробка* ***(refactoring)*** - системи для усунення зайвої складності, збільшення зрозумілості коду, підвищення його гнучкості. При цьому перевага віддається більш елегантним і гнучким рішенням, у порівнянні з тими, які просто дають потрібний результат.
  + *Програмування парами* ***(pair programming)*** - весь код пишеться двома програмістами на одному комп'ютері, що підвищує його якість (відсутність помилок, зрозумілість, читання, ...).
  + *Постійна інтеграція* ***(continuous integration)*** - система збирається і проходить інтеграційне тестування якомога частіше, по кілька разів на день кожного разу, коли пара програмістів закінчує реалізацію чергової функції.
  + ***40-****годинний робочий тиждень* - понаднормова робота розглядається як ознака великих проблем у проекті. Не допускається понаднормова робота 2 тижні поспіль - це виснажує програмістів і робить їх роботу значно менш продуктивною.

**Гнучке розроблення ПЗ на основі Agile. Архітектура ПЗ. Стандарти опису архітектури. Шаблони проектування. Патерни.**

Гнучке розроблення ПЗ на основі **Agile**

Постійна зміна вимог під час розроблення ПЗ, необхідність забезпечення ефективної співпраці команди розробників потребували методів, які б забезпечили якість розроблення та супроводу програмних систем і уникнули недоліків занадто формалізованих методів, більшість з яких базувалась на каскадній моделі ЖЦ.

У 90-х рр. ХХ ст. активно стали виникати різноманітні підходи до створення ПЗ, які забезпечували гнучку роботу програмістів.

У результаті у 2001 році був написаний *Маніфест гнучкого розроблення* ***(Agile manifesto)***, що зафіксував цінності ефективних підходів до розроблення програмних продуктів, таких, як XP, Feature driven development, Scrum, Adaptive software development, Pragmatic Programming (прагматичне програмування).

Текст маніфесту підписали 17 найавторитетніших фахівців у цій галузі діяльності – Кент Бек (Kent Beck), Алістер Коуберн (Alistair Cockburn), Мартін Фаулер (Martin Fowler) та інші:

***Agile-****маніфест розроблення програмного забезпечення*

*Ми постійно відкриваємо для себе досконаліші методи розроблення програмного забезпечення, займаючись розробленням безпосередньо та допомагаючи у цьому іншим. Завдяки цій роботі ми змогли зрозуміти, що:*

* *Люди та співпраця важливіші за процеси та інструменти;*
* *Працюючий продукт* важливіший за вичерпну документацію;
* *Співпраця із замовником* важливіша за обговорення умов контракту;
* *Готовність до змін* важливіша за дотримання плану.

Хоча цінності, що справа, важливі, ми все ж цінуємо більше те, що зліва.

У результаті з’явився термін – *Гнучке розроблення програмного* *забезпечення* ***(Agile software development)*** – клас методологій розроблення програмного забезпечення, що базуються на ітеративному розробленні, в якому вимоги та рішення еволюціонують через співпрацю між самоорганізовуваними багатофункціональними командами.

*У маніфесті озвучені основні принципи гнучкого розроблення ПЗ:*

* *Найвищий пріоритет* – задоволення потреб замовника шляхом завчасного та регулярного постачання програмного забезпечення.
* *Схвальне ставлення до змін*, навіть на завершальних стадіях розроблення. Agile-процеси використовують зміни для забезпечення конкурентоспроможності замовника.
* *Працюючий продукт* необхідно випускати якомога частіше, з періодичністю від двох тижнів до двох-трьох місяців.
* Упродовж усього проекту *розробники і представники бізнесу повинні працювати разом щодня*.
* *Над проектом повинні працювати вмотивовані професіонали*, які працюють у зручних умовах, із повною підтримкою та довірою менеджменту проекту.
* *Особиста комунікація* – найефективніший та найпрактичніший метод обміну інформацією в команді.
* *Працюючий продукт* – головний показник прогресу.
* Інвестори, розробники і користувачі повинні мати можливість підтримувати *постійний ритм роботи*. Agile допомагає налагодити такий сталий процес розроблення.
* *Постійна увага* до технічної досконалості і якості проектування підвищує гнучкість проекту.
* *Простота* – мистецтво мінімізації зайвої роботи – дуже необхідна.
* Найкращі вимоги, архітектурні та технічні рішення виникають у командах, які здатні *самоорганізовуватись.*
* *Команда постійно шукає способи підвищення ефективності* та відповідно коригує свою роботу.

Більшість гнучких методологій націлені на мінімізацію ризиків шляхом зведення розроблення до серії коротких циклів – ітерацій, які, як правило, тривають один-два тижні. Кожна ітерація є програмним проектом у мініатюрі і включає всі завдання, необхідні для отримання мінімального приросту функціональності: планування, аналіз вимог, проектування, кодування, тестування та документування.

*Гнучкий програмний проект* передбачає в кінці кожної ітерації отримання працездатного, готового до встановлення у реальному середовищі продукту. Після закінчення кожної ітерації команда виконує переоцінку пріоритетів розроблення.

*Гнучкі методи* орієнтовані на різні аспекти життєвого циклу розроблення програмного забезпечення. Деякі акцентують увагу на практичних завданнях (екстремальне програмування, прагматичне програмування, Agile-моделювання), інші зосереджені на управлінні програмними проектами (Scrum). Також серед agile-методів існують підходи, які забезпечують повне охоплення життєвого циклу розроблення – метод розроблення динамічних систем (dynamic systems development method, DSDM) та уніфікований процес розроблення (RUP), розглянутий вище. Більшість гнучких методів для використання упродовж життєвого циклу ПЗ потрібно доповнювати іншими підходами, окрім DSDM та RUP.

Особливістю гнучких методів є те, що на даний час відсутні дані про провальні agile-проекти, але є результати опитувань, які підтвердили їх успішне використання. Тому і потужні компанії-розробники активно їх запроваджують у свою діяльність, наприклад, Oracle запроваджує гнучкі методи управління ЖЦ продуктів (Agile product lifecycle management), фірма IBM є власником продуктів, які підтримують використання методології RUP.

*Потрібно виділити фактори, які можуть негативно вплинути на використання гнучких методів:*

* + масштабні зусилля у галузі розвитку (більше 20 розробників);
  + розподілена у просторі команда;
  + примусове впровадження гнучкого процесу усупереч вимогам команди розробників;

Небажане використання agile-методів у критично важливих системах, де відмова ПЗ неможливий ні в якому разі (наприклад, управління повітряним рухом).

Agile акцентує увагу на безпосередньому спілкуванні між учасниками проекту. Більшість agile-команд розміщені в одному офісі (bullpen). До команди обов’язково включають представників замовника (замовники, які визначають продукт, менеджери продукту, бізнес-аналітики або користувачі). Також до команди входять тестувальники, дизайнери інтерфейсу, технічні автори та менеджери.

Із agile-методів потрібно виділити Scrum, що встановлює правила керування процесом розроблення та дозволяє використовувати вже існуючі практики кодування, коригуючи вимоги або вносячи тактичні зміни. Використання цієї методології дає можливість виявляти і усувати відхилення від бажаного результату на найбільш ранніх етапах розроблення програмного продукту.

Scrum-методи ітераційні та інкрементні. Кожна ітерація (спринт) триває упродовж 15-30 днів і повинна закінчитися приростом функціональних можливостей.

Під час спринту обов’язково відбуваються зустрічі дійових осіб:

* + *Планування спринту* ***(Planning Meeting)*** – відбувається на початку ітерації. Не більше 4-8 годин.
  + *Мітинг* ***(Daily Scrum)*** – відбувається кожного дня упродовж спринту. Триває 15 хвилин.
  + *Демонстрація* ***(Demo Meeting)*** – відбувається в кінці ітерації (спринту). Обмежена 4 годинами.
  + *Ретроспектива* ***(Retrospective Meeting)***. Усі члени команди розповідають про своє ставлення до ходу спринту. Обмежено 1-3 годинами.

*Дійові особи розподіляються на дві групи:*

*повністю задіяні* у процесі розроблення («свині»):

* *Власник Продукту* ***(Product Owner);***
* *Керівник* ***(ScrumMaster);***
* *Команда* ***(Scrum Team);***

*причетні* до розроблення («кури»):

* *Користувачі* ***(Users);***
* *Клієнти****,*** *Продавці* ***(Stakeholders);***
* *Експерти****-****консультанти* ***(Consulting Experts).***

Віддаючи перевагу безпосередньому спілкуванню, agile-методи зменшують обсяг письмової документації порівняно з іншими методами, що викликає критику прихильників більш формалізованих підходів.

**Архітектура ПЗ. Стандарти опису архітектури**

*Архітектура програми* включає в себе найбільш важливі статичні і динамічні аспекти системи та будується на основі вимог до результату. Як було відмічено вище, вимоги відбиваються у варіантах використання. Однак прецеденти також залежать від безлічі інших чинників, таких, як вибір платформи для роботи програми (тобто комп'ютерної архітектури, операційної системи, СКБД, мережевих протоколів), доступність готових блоків багаторазового використання (наприклад, каркасу GUI), спосіб розгортання, успадковані системи і нефункціональні вимоги (наприклад, продуктивність і надійність).

*Архітектура ПЗ* – це представлення всього проекту із виділенням важливих характеристик без акценту на деталях.

Кожен продукт має функції і форму. Одне без іншого не існує. Функції відповідають варіанту використання, а форма – архітектурі. В реальних умовах архітектура і прецеденти розробляються паралельно. Архітектура повинна бути спроектована так, щоб дозволити системі розвиватися не тільки в момент початкового розроблення, але і в майбутніх поколіннях.

Щоб знайти таку форму, архітектор повинен працювати, повністю розуміючи ключові функції, тобто ключові варіанти використання системи. Ці ключові варіанти використання становлять 5-10% усіх ВВ, але вони дуже важливі, оскільки містять функції ядра системи.

*Архітектор виконує такі кроки:*

* + *визначає платформу системи* – створює грубий ескіз архітектури, починаючи з тієї частини, що не пов'язана з варіантами використання. Хоча ця частина архітектури не залежить від прецедентів, архітектор повинен у загальних рисах розуміти варіанти використання до створення ескізу архітектури;
  + далі архітектор *працює із підмножиною виділених ВВ*, кожен з яких відповідає одній із ключових функцій розроблюваної системи. Кожен з обраних прецедентів детально описується і реалізується в поняттях підсистем, класів і компонентів;
  + після того як ВВ описані та повністю розроблені, більша частина архітектури досліджена. Створена архітектура, у свою чергу, буде базою для повного розроблення інших ВВ. Цей процес продовжується до того часу, поки архітектура не буде визнана стабільною.

**Шаблони проектування. Патерни**

Створюючи об’єкт, у тому числі й програмний продукт, розробник часто стикається із завданнями, які вже хто-небудь вирішив. У 70-ті рр. ХХ ст архітектор Кристофер Александер (Christopher Alexander) запропонував шаблони проектування будинків та міст. Через десятиліття ця ідея переросла у процесі розроблення ПЗ до шаблонів проектування інтерфейсу користувача, запропонованих Вардом Каннінгемом (Ward Cunningham) та Кентом Беком.

Далі ідея була активно підхоплена та розвинута у вигляді каталогу патернів ООП. Цей каталог став дуже популярним серед розробників і часто згадується як патерни GoF («Gang of Four», або «банда чотирьох», – за кількістю авторів). Ідея повторного використання не тільки коду, а й архітектурних та проектних рішень виявилася настільки успішною, що сьогодні патерни проектування широко застосовуються в різних методиках розроблення програмного забезпечення.

У роботі під *патернами проектування* об’єктно-орієнтованих систем розуміють опис взаємодії об’єктів і класів, адаптованих для вирішення спільної задачі проектування в конкретному контексті.

Існує багато патернів розроблення програмних систем, які відмінні сферою застосуання, масштабом, змістом, стилем опису. Наприклад, залежно від області використання існують такі патерни, як патерни аналізу, проектування, кодування, рефакторингу, тестування, реалізації корпоративних застосувань, шаблони роботи з базами даних, шаблони розподілених додатків, шаблони роботи із багатопотоковістю, шаблони документування та ін. В останнє десятиліття шаблони впроваджені навіть у роботу менеджера процесу розроблення ПЗ (Джеймса Коплі, Ніла Харрісона «Organizational Patterns of Agile Software Development», Тома Демарко, Тіма Лістера «Adrenaline Junkies and Template Zombies: Understanding Patterns of Project Behavior»).

У даний час найбільш популярними *патернами є патерни проектування****.*** Однією з найпоширеніших класифікацій таких патернів є класифікація за ступенем деталізації та рівнем абстракції розглянутих систем.

*Патерни проектування програмних систем поділяються на три категорії* (рис. 9.1).

*Архітектурні патерни*, найбільш високорівневі, описують структурну схему програмної системи в цілому. У даній схемі зазначаються окремі функціональні складові системи (підсистеми), а також взаємовідносини між ними. Прикладом архітектурного патерну є добре відома програмна парадигма "модель-вигляд-контролер" (model-view-controller – MVC).

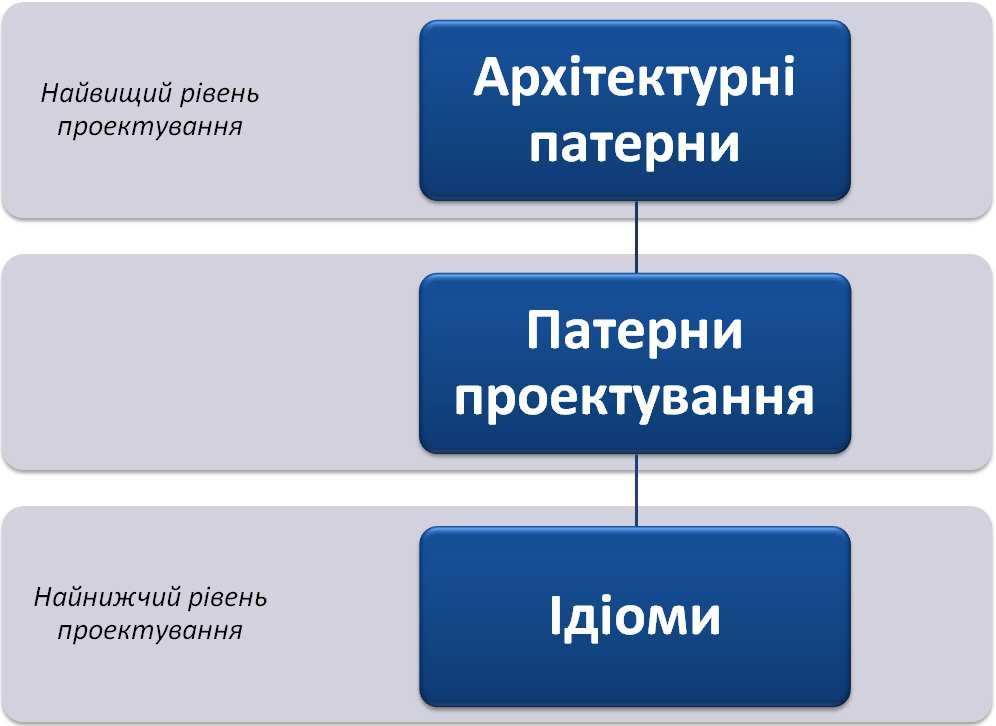


Рисунок 9.1 – Шаблони проектування програмних систем

*Ідіоми*, низькорівневі патерни, мають справу з питаннями реалізації певної проблеми з урахуванням особливостей мови програмування. При цьому часто одні й ті самі ідіоми для різних мов програмування мають різний вигляд або взагалі відсутні. Наприклад, в C++ для усунення можливих втрат пам'яті можуть використовуватися інтелектуальні покажчики. Інтелектуальний покажчик містить покажчик на ділянку динамічно виділеної пам'яті, що буде автоматично звільнений при виході із зони видимості. У середовищі Java такої проблеми просто не існує, оскільки там використовується автоматичне складання сміття. Як правило, для використання ідіом потрібно глибоко знати особливості застосовуваної мови програмування.

*Завдання кожного патерну* – дати чіткий опис проблеми та її вирішення у відповідній області. У загальному випадку опис патерну завжди містить такі елементи:

* *Назва патерну* – унікальне змістове ім'я, що однозначно визначає дану задачу або проблему і її рішення.
* *Розв****'****язувана задача* – надається розуміння того, чому розв'язувана проблема дійсно є такою, чітко описує її межі.
* *Рішення* – зазначається, як саме дане рішення пов'язане з проблемою, наводяться шляхи її вирішення.
* *Результати використання патерну* – як правило, це переваги, недоліки та компроміси.

На рисунку 9.2 показана класифікація шаблонів проектування інформаційних систем.

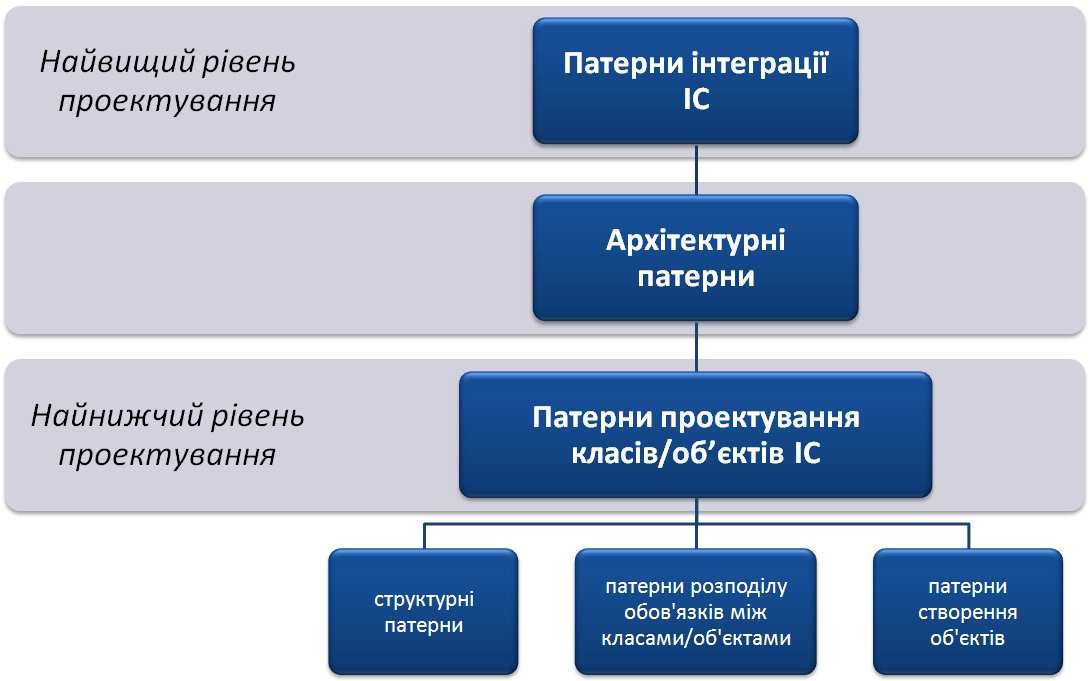


Рисунок 9.2 – Шаблони проектування інформаційнх систем

*Архітектурні патерни також об’єднуються у групи:*

* *структурні, архітектурні патерни* – слугують для організації класів або об’єктів системи у базовій підструктурі;
* *патерни управління* – для забезпечення необхідного системного функціоналу.

У свою чергу, патерни управління розділені на патерни централізованого управління (патерни, в яких одна з підсистем повністю відповідає за управління, запускає і завершує роботу решти підсистем), патерни управління, які передбачають децентралізоване реагування на події (згідно з цим патернам на зовнішні події відповідає відповідна підсистема), та патерни, які описують організацію зв'язку з базою даних.

Патерни інтеграції інформаційних систем знаходяться на верхньому рівні класифікації патернів проектування. Аналогічно патернам більш низьких рівнів класифікації серед патернів інтеграції виділена група структурних патернів. Структурні патерни описують основні компоненти інтегрованої метасистеми. Для опису взаємодії окремих корпоративних систем, включених до інтегрованої метасистеми, використовується група патернів, об'єднаних відповідно до методу інтеграції. Інтеграція корпоративних інформаційних систем передбачає організований обмін даними між системами, для якого використовується відповідний патерн. Необхідно зазначити, що на відміну від патернів проектування класів/об’єктів і архітектурних системних патернів віднесення окремого патерну інтеграції до того чи іншого виду є менш умовним.

Про шаблони проектування знають розробники, проектувальники, архітектори, але про це явище абсолютно нічого не відомо користувачам, для яких і розроблялася мова шаблонів. На сьогоднішній день основна роль шаблонів – це повторне використання досвіду в різних областях розроблення ПЗ, усунення комунікаційного бар'єру всередині команди розробників і між ними, підвищення якості створюваного продукту за рахунок використання перевірених роками рішень.