Лекція 2

Процес розробки програмного забезпечення

Анотація: Поняття процесу розробки ПЗ. Універсальний процес. Поточний процес. Конкретний процес. Стандартний процес. Вдосконалення процесу. Pull/push стратегії. Класичні моделі процесу: модель водоспаду, спіральна модель. Фази і види діяльності.

Процес

Як ми працюємо, яка послідовність наших кроків, які *норми* і правила в поведінці і роботі, який регламент відносин між членами команди, як проект взаємодіє із зовнішнім світом і т.і.? Все це разом ми схильні називати процесом. Його усвідомлення, вибудовування і *поліпшення* – основа будь-якої ефективної групової діяльності. Тому не випадково, що процес виявився одним з основних понять *програмної інженерії*.

Центральним об'єктом вивчення *програмної інженерії* є **процес** створення ПЗ – безліч різних видів діяльності, методів, методик і кроків, що використовуються для розробки і *еволюції ПЗ* і пов'язаних з ним продуктів (проектних планів, документації, програмного коду, тестів, призначеної для користувача документації та ін.).

Проте, на сьогоднішній день не існує **універсального процесу** розробки ПЗ – набору методик, правил і розпоряджень, які б відповідали б ПЗ будь-якого виду, для будь-яких компаній, для команд будь-якої національності. Кожен **поточний процес** розробки, який здійснюється деякою командою в межах *певного проекту*, має велику кількість особливостей і індивідуальностей. Проте доцільно перед початком проекту спланувати процес роботи, визначивши ролі і обов'язки в команді, *робочі продукти* (проміжні і фінальні), порядок участі в їх розробці членів команди і так далі. Називатимемо цей попередній опис **конкретним процесом**, відрізняючи його від плану *робіт*, проектних специфікацій і ін. Наприклад, в системі Microsoft *Visual Team System* опиняється *шаблон* процесу, що створюється або адаптується (у разі використання стандартного) перед початком розробки. У VSTS існують заготовки для конкретних процесів на базі *CMMI*, Scrum і ін.

В межах окремої компанії можлива і корисна стандартизація всіх поточних процесів, яку називатимемо **стандартним процесом**. Останній, таким чином, виявляється деякою базою даних, що містить такі дані:

- інформацію, правила використання, документацію і інсталяційні пакети *засобів розробки*, які використовуються в

проектах компанії (систем версійного контролю, засобів контролю помилок, засобів програмування – різних *IDE*, *СУБД* і так далі);
- опис практик розробки – проектного *менеджменту*, правил роботи із замовником і т.д.;
- шаблони проектних документів – *технічних завдань*, проектних специфікацій, планів тестування і так далі і ін.

Також можлива стандартизація процедури розробки конкретного процесу як частки стандартного. Основна ідея стандартного процесу – курсування усередині компанії передового досвіду, а також *уніфікація засобів розробки*. Дуже вже часто в компаніях різні департаменти і проекти сильно відрізняються *за зрілістю процесу розробки*, що ускладнює повторне використання передового досвіду. Крім того, трапляється, що компанія використовує декілька засобів паралельних інструментів розробки, наприклад, *СУБД* засобу версійного контролю. Іноді це буває виправдано (наприклад, такі вимоги *замовника*), часто це необхідно, наприклад, *Java .NET* (компетентність офшорної компанії дозволяє їй обирати ширший спектр замовлень). Але дуже часто це довільний вибір самих розробників. У будь-якому випадку, така множинність істотно ускладнює міграцію фахівців з проекту в проект, використання результатів одного проекту в іншому і так далі. Проте, у разі організації стандартного процесу необхідно стежити, щоб стандартний процес не виявився лише формальним, бюрократичним апаратом. Поняття стандартного процесу введене і детально описане в підході *CMMI*.

Необхідно відзначити, що наявність стандартного процесу свідчить про наявність "єдиної волі" в організації, яка існує саме на рівні процесу. На рівні продажу, бухгалтерії та інших звичних для всіх компаній процесів і *активів* єдність здійснити не важко. А ось на рівні процесів розробки дуже часто кожен проект виявляється сам *по* собі (особливо в офшорних проектах) –буденність захоплює і ізолює проекти один від одного дуже міцно.

Вдосконалення процесу

Визначення. Вдосконалення процесу (*software process improvement*) – це *діяльність щодо* змін існуючого процесу (як поточного, в межах одного проекту, так і стандартного, для всієї компанії) з метою поліпшення якості створюваних продуктів і/або зниження ціни і часу їх розробки. Причини актуальності цієї діяльності для компаній-виробників полягає в наступному.

1. Відбувається швидка зміна технологій розробки ПЗ, потрібні вивчення і впровадження нових засобів розробки.
2. Спостерігається швидке зростання компаній і їх вихід на нові ринки, що вимагає нової організації робіт.

3. Має місце висока конкуренція, яка вимагає пошуку ефективніших, більш економічних способів розробки.

Що і яким чином можна покращувати.

1. Перехід на нові засоби розробки, *мови програмування* і так далі.
2. *Поліпшення* окремих управлінських і інженерних практик – тестування, *управління вимогами* і ін.
3. Повна, комплексна перебудова всіх процесів в проекті, департаменті, компанії (у відповідності, наприклад, до *CMMI*).
4. Сертифікація компанії (*CMM/CMMI*, *ISO 9000* і ін.).

Ми відокремили п. 3 від п. 4 тому, що на практиці 4 далеко не завжди означає дійсну творчу роботу *із* поліпшення процесів розробки *ПЗ,* а часто зводиться до підтримки відповідного документообігу, який є необхідним для отримання сертифікації. Сертифікат потім використовується як засіб, козир в боротьбі за замовлення.

Головна важкість реального вдосконалення процесів в компанії полягає в тому, що вона при цьому повинна працювати і створювати *ПЗ,* її не можна "закрити на переоблік".

Звідси витікає ідея безперервного поліпшення процесу, так би мовити, малими порціями, щоб процедура була не такою хворобливою. Це є тим більш логічним, що нові технології розробки, які з'являються на ринку, а також розвиток тих, що вже існують потрібно постійно відслідковувати. Ця стратегія, зокрема, отримала віддзеркалення в стандарті вдосконалення процесів розробки *CMMI*.

Pull/push стратегії. У контексті впровадження *інновацій* у виробничі процеси бізнес-компаній (не обов'язково компанії для створення *ПЗ*) існують дві *парадигми*.

1. Organization pull – інновації, які націлені на вирішення конкретних проблем компанії.

2. Technology push – широкомасштабне впровадження *інновацій* із стратегічних міркувань. Замість конкретних проблем, які будуть вирішені після впровадження *інновації,* в цьому випадку розглядаються показники компанії (ефективність, *продуктивність*, річний оборот засобів, збільшення *вартості акцій* публічної компанії), які будуть збільшені, покращені після впровадження *інновації*. При цьому передбачається, що будуть автоматично вирішені численні окремі проблеми організації, у тому числі і ті, про які в даний момент нічого не відомо.

Приклад використання стратегії *organization pull* – впровадження нових засобів тестування за ситуації, коли є високі вимоги *до якості* в проекті, або коли якість програмної системи не задовольняє замовника.

Приклад використання стратегії *technology push* – перехід компанії із засобів структурної розробки на об’єктно-орієнтоване програмування. Ще один приклад використання тієї ж стратегії – впровадження стандартів якості ISO *9000* або CMMI. У обох цих випадках компанія не вирішує якусь одну проблему або ряд проблем – вона хоче радикально змінити ситуацію, вийти на нові позиції і так далі.

Проблеми застосування стратегії *technology push* полягають в тому, що потрібна глобальна перебудова процесу. Але компанію не можна "закрити на реконструкцію" – за цей час місце компанії на ринку може бути зайняте конкурентами, акції компанії можуть впасти і так далі. Таким чином, впровадження *інновацій*, як правило, відбувається паралельно із звичайною діяльністю компанії, поетапно, що у випадку з *technology push* зв'язано з великими труднощами і ризиками.

Використання стратегії *organization pull* менш ризиковане, зміни, що вносяться нею в процес, менш глобальні, локальні. Але і переваг такі *інновації* приносять менше, ***в порівнянні з*** вдалими впровадженням відповідно до стратегії *technology push*.

Необхідно також відзначити, що існують проблеми, які неможливо усунути точковими переробками процесу, тобто необхідно застосовувати стратегію *technology push*. Наведемо процес супроводу і розвитку сімейства *програмних продуктів*, який, наприклад, зайшов у *безвихідь,* – компанія стає збитковою, якщо і далі буде супроводжувати встановлені раніш продукти, інструментальні засоби проекту безнадійно застаріли і знаходяться в жалюгідному стані, *менеджмент* засмучений, всі спроби керівництва змінити процес натрапляють на нерозуміння колективу, сварки і *конфлікти*. Можливо, що у такому разі без "революції" не обійтися.

Ще одна відмінність обох стратегій: у випадку з *organization pull*, як правило, повернення інвестицій від впровадження відбувається швидше, ніж у випадку з *technology push*.

Класичні моделі процесу

Визначення моделі процесу. Процес створення *програмного забезпечення* не є однорідним. Той або інший метод розробки *ПЗ,* як правило, визначає деяку динаміку *розгортання* тих або інших видів діяльності, тобто, визначає модель *процесу* (*process model*).

Модель є гарною *абстракцією* різних методів розробки *ПЗ,* яка дозволяє лаконічно, стисло і інформативно їх унаочнити. Проте, сама ідея *моделі процесу* є однією з найбільш ранніх в *програмній інженерії*, коли вважалося, що вдала модель – це найголовніше, що сприяє успіху розробки. Пізніше прийшло усвідомлення, що існує безліч інших аспектів (принципи управління і розробки, структура команди і так далі), які повинні бути узгоджені один з одним. І тоді почали розвиватися

інтегральні *методології розробки*. Проте, існує декілька класичних моделей процесу, які корисні на практиці і на яких слід зупинитись.

Фази і види діяльності. Кажучи про моделі процесів, необхідно розрізняти фази і види діяльності.

Фаза (*phase*) – це певний етап процесу, який має початок, кінець і вихідний результат. Наприклад, фаза перевірки здійсненності проекту, здача проекту і так далі. Фази слідують одна за одною в лінійному порядку, характеризуються наданням звітності замовникові і, часто, виплатою грошей за виконану частину роботи.

Поодинокий замовник погодиться перший раз побачити результати лише після завершення проекту. З іншого боку, підрядчики вважають за краще отримувати гроші поступово, *у міру того, як* виконуються окремі частини роботи. Таким чином, з'являються фази, що дозволяють створювати і пред'являти проміжні результати проекту. Фази корисні також безвідносно взаємодії із замовником – з їх допомогою можна синхронізувати *діяльність* різних *робочих груп*, а також відслідковувати перебіг проекту. Прикладами фаз може служити узгодження із замовником технічного завдання, реалізація певної функціональності *ПЗ,* етап розробки, що закінчується здачею системи на тестування або випуском альфа-версії.

Вид діяльності (*activity*) – це певний тип роботи, що виконується в процесі розробки *ПЗ.* Різні види діяльності часто вимагають різних професійних навичок і виконуються різними фахівцями. Наприклад, *управління проектом виконується менеджером проекту*, *кодування – програмістом, тестування – тестувальником*. Є види діяльності, які можуть виконуватися одними і тими самими фахівцями, – наприклад, *кодування* і *проектування* (особливо в невеликому проекті) часто виконують одні і ті ж самі люди.

В межах однієї фази може виконуватися багато різних видів діяльності. Крім того, один вид діяльності може виконуватися на різних фазах – наприклад, тестування: на фазі аналізу і проектування можна писати тести і налагоджувати тестове оточення, а саме тестування проводити під час розробки і перед здачею. На поточний момент для складного *програмного забезпечення* використовуються *багатовимірні моделі* процесу, в яких відокремлення фаз від видів діяльності істотно полегшує управління розробкою *ПЗ.*

Види діяльності, фактично, присутні, під різними назвами, в кожному методі розробки *ПЗ.* У *RUP* вони мають назву робочі *процеси* (*work flow*), в *CMM* – ключовими галузями процесу (key *process area*). Слід зберігати традиційні назви, які прийняті в тому або іншому методі, щоб не створювати плутанини.

Модель водоспаду була запропонована в 1970 році Вінстоном Ройсом. Фактично, вперше в процесі розробки ПЗ були виділені різні етапи розробки і змінені примітивні уявлення про розробку ПЗ у вигляді аналізу системи і її кодування.

Були визначені такі етапи: розробка *системних вимог*, розробка вимог до *ПЗ, аналіз*, проектування, *кодування*, тестування, експлуатація і супровід (**Рис. 2.1)**.

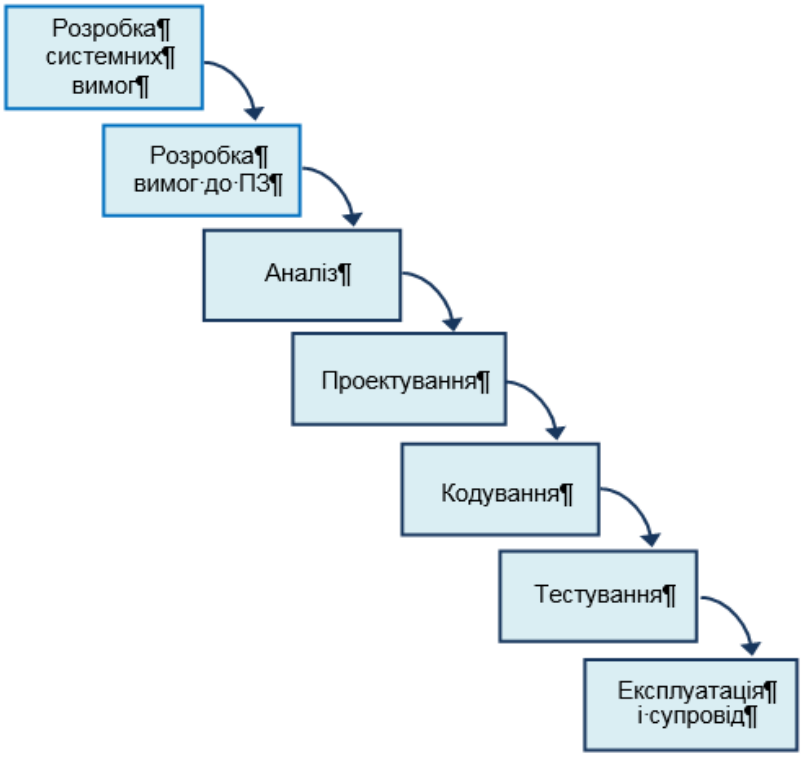

Рис. 2.1. Модель водоспаду

Іноді каскадну модель зображають з іншою послідовністю етапів розробки ПЗ (**Рис. 2.2)**.

Рис. 2.2. Альтернативна модель водоспаду

Перевагою цієї моделі є обмеження можливості повернення на довільну кількість кроків назад, наприклад, від тестування – до аналізу, від розробки – до роботи над вимогами і так далі. Наголошувалося, що такі повернення можуть катастрофічно збільшити вартість проекту і терміни його виконання. Наприклад, якщо при тестуванні виявляються помилки проектування або аналізу, то їх виправлення часто призводить до повної переробки системи. Цією моделлю допускалися повернення лише на попередній крок, наприклад, від *тестування* до *кодування*, від *кодування* до *проектування* і так далі.

Нарешті, в межах цієї моделі було введено прототипіювання, тобто пропонувалося розробляти систему двічі, аби зменшити ризики помилок підчас розробки. Перша версія – прототип – дозволяє побачити основні ризики і обґрунтовано ухвалити головні архітектурні рішення. На створення прототипу відводилося до однієї третини часу всієї розробки.

У 70-80 роках минулого століття ця модель через свою простоту і схожість з моделями розробки інших, не програмних, систем широко використовувалася в процесі розробки ПЗ. Надалі, у зв'язку з розвитком *програмної інженерії* і усвідомленням ітеративного характеру процесу розробки ПЗ, ця модель активно критикувалася, практично, кожним автором відповідних статей і підручників. Стала загальноприйнятою думка, що вона не відображає особливостей розробки *ПЗ.* Недоліками моделі водоспаду є:

- ототожнення фаз і видів діяльності, що призводить до втрати гнучкості розробки, зокрема, до труднощів підтримки ітеративного процесу розробки;
- вимога повного закінчення фази-діяльності, закріплення результатів у вигляді докладного відповідного документа (технічного завдання, проектної специфікації); проте досвід розробки ПЗ показує, що неможливо цілком завершити розробку вимог, дизайн системи і так далі – все це вимагає змін; і причини тут не лише в тому, що оточення проекту є змінним, але і в тому, що заздалегідь важко точно визначити і сформулювати багато рішень, вони з’ясовуються і деталізуються лише згодом;
- інтеграція всіх результатів розробки відбувається в кінці проекту, унаслідок чого інтеграційні проблеми дають про себе знати занадто пізно;
- користувачі і замовник не можуть ознайомитися з варіантами системи під час розробки, а бачать результат лише в самому кінці; тим самим, вони не можуть вплинути на процес створення системи, і тому збільшуються ризики непорозуміння між розробниками і користувачами або замовником;

- модель є нестійкою до збоїв у фінансуванні проекту або перерозподілу грошових коштів, розробка яку розпочали, фактично, не має альтернатив "за перебігом справи".

Проте дана модель продовжує використовуватися на практиці – для невеликих проектів або при розробці типових систем, де ітеративність не є важливою. З її допомогою зручно відстежувати розробку і здійснювати поетапний *контроль* за проектом. Ця модель також часто використовується в офшорних проектах з почасовою оплатою праці. Модель водоспаду увійшла як складова частина в інші моделі і методології, наприклад, в *MSF*.

Спіральна модель була запропонована Бері Боемом в 1988 році для подолання недоліків моделі водоспаду, перш за все, для кращого управління ризиками. Згідно цієї моделі розробка продукту здійснюється у вигляді спіралі, кожен виток якої є певною фазою розробки. На відміну від моделі водоспаду в спіральній моделі немає зумовленого і обов'язкового набору витків, кожен виток може стати останнім підчас розробки системи, а після його завершення складаються плани наступного витка. Нарешті, виток є саме фазою, а не видом діяльності, як в моделі водоспаду, в його межах може здійснюватися багато різних видів діяльності, тобто модель є двовимірною (Рис. 2.3).

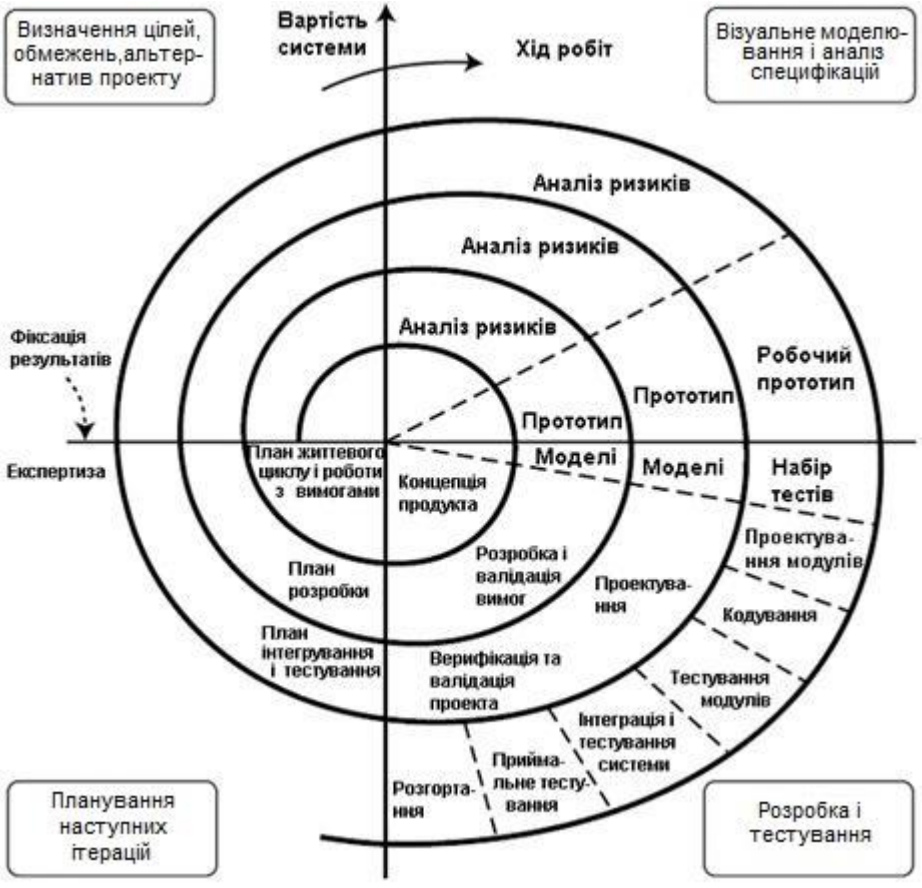

Рис. 2.3. Модель спірального процесу розробки ПЗ

Послідовність витків може бути такою: на першому витку ухвалюється рішення про доцільність створення *ПЗ,* на наступному визначаються *системні вимоги*, потім здійснюється проектування системи і так далі. Витки можуть мати і інші значення.

Кожен виток має таку структуру (сектори):

- визначення цілей, обмежень і альтернатив проекту;
- оцінка альтернатив, оцінка і дозвіл ризиків; можливе використання прототипіювання (зокрема створення серії прототипів), симуляція системи, візуальне моделювання і аналіз специфікацій; фокусування на самих ризикових частинах проекту;
- розробка і тестування – тут можлива модель водоспаду або використання інших моделей і методів розробки ПЗ;
- планування наступних ітерацій – аналізуються результати, плани і ресурси для подальшої розробки, ухвалюється (або не приймається) рішення про новий виток; аналізується, чи має сенс продовжувати розробляти систему чи ні; розробку можна і припинити, наприклад, через збій у фінансуванні; спіральна модель дозволяє зробити це коректним чином.

Окрема спіраль може відповідати розробці деякою програмною компоненти або внесенню чергових змін до продукту. Таким чином, у моделі може з'явитися третій вимір.

Спіральну модель недоцільно застосовувати в проектах з невеликим ступенем ризику, з обмеженим бюджетом, для невеликих проектів. Крім того, відсутність хороших засобів прототипіювання може також зробити незручним використання спіральної моделі.

Спіральна модель не знайшла широкого застосування в індустрії і важлива, швидше, лише в історико-методологічному плані: вона є першою ітеративною моделлю, має красиву метафору – спіраль, – і, подібно до моделі водоспаду, застосовувалася надалі у випадку створення інших моделей процесу і методології розробки *ПЗ.*

Основні етапи життєвого циклу системи

- Визначення мети або опис проблем.
- Розробка вимог та критеріїв.
- Синтез системних рішень.
- Аналіз системних рішень. Валідація проектних рішень
- Вибір системи.
- Реалізація системи.

Вимоги до системи в інженерії

Всі вимоги до будь-якої системи можна розбити на три категорії (або класи):

- **Основні вимоги**. Це такі вимоги, які виконавець отримує від замовників та інших зацікавлених сторін. Вони можуть бути просторовими і загальними, деталізованими і специфічними, повними і фрагментарними – у більшості випадків, в них є всього потроху. Іноді прийнято говорити: "Коли справа доходить до вимог, замовник не приймає ніяких правил - ви отримуєте те, що отримуєте".
- **Цільові або бізнес-вимоги**. Це вимоги, що визначають контекст, в якому система буде функціонувати, – але не те, що саме вона робить, а та роль, яку вона відіграє в глобальному вимірі. Скажімо, для проектованого квадрокоптера це буде опис місій, для яких він буде призначатися. Для нового смартфона бізнес-вимоги можуть визначати, яким чином він буде функціонувати в інфраструктурі конкретного мобільного оператора.
- **Системні/підсистемні вимоги**. Ці вимоги визначають те, що система повинна мати можливість робити. Вони беруть свій початок на високому системному рівні, потім аналізуються і декомпозуються, щоб створити вимоги для підсистем рівнем нижче (Рис. 2.4). Вони можуть бути виражені у простих формах ("система повинна ..."), або зображуватися у вигляді моделей і діаграм.

V-модель процесу системної інженерії

Модель процесу системної інженерії наведена на Рис. 2.4.

Підчас переміщення V-моделлю зліва направо, процес системної інженерії реалізується у вигляді серії певних кроків:

- Реалізується концепція використання (Conceptual Options): визначаються і інвентаризуються потреби основних зацікавлених сторін, загальні можливості системи, ролі і обов'язки, а також вимірювані показники ефективності, яким система повинна відповідати під час її валідації після закінчення проекту.
- Визначаються характеристики системи: розробляється набір системних вимог, які піддаються перевірці та відповідають потребам зацікавлених сторін, що були визначені на стадії розробки і прийняття концепції використання системи.
- Відбувається високорівневе проектування: розробляється високорівнева архітектура, яка задовольняє системним вимогами, забезпечує обслуговування, можливу модернізацію, а також інтеграцію з іншими системами.
- Реалізується проектування компонентів: поступово деталізується системний дизайн, формуються такі вимоги до компонентів, що дозволять купувати і використовувати

апаратні засоби, вартість яких не буде виходити за межі бюджету.

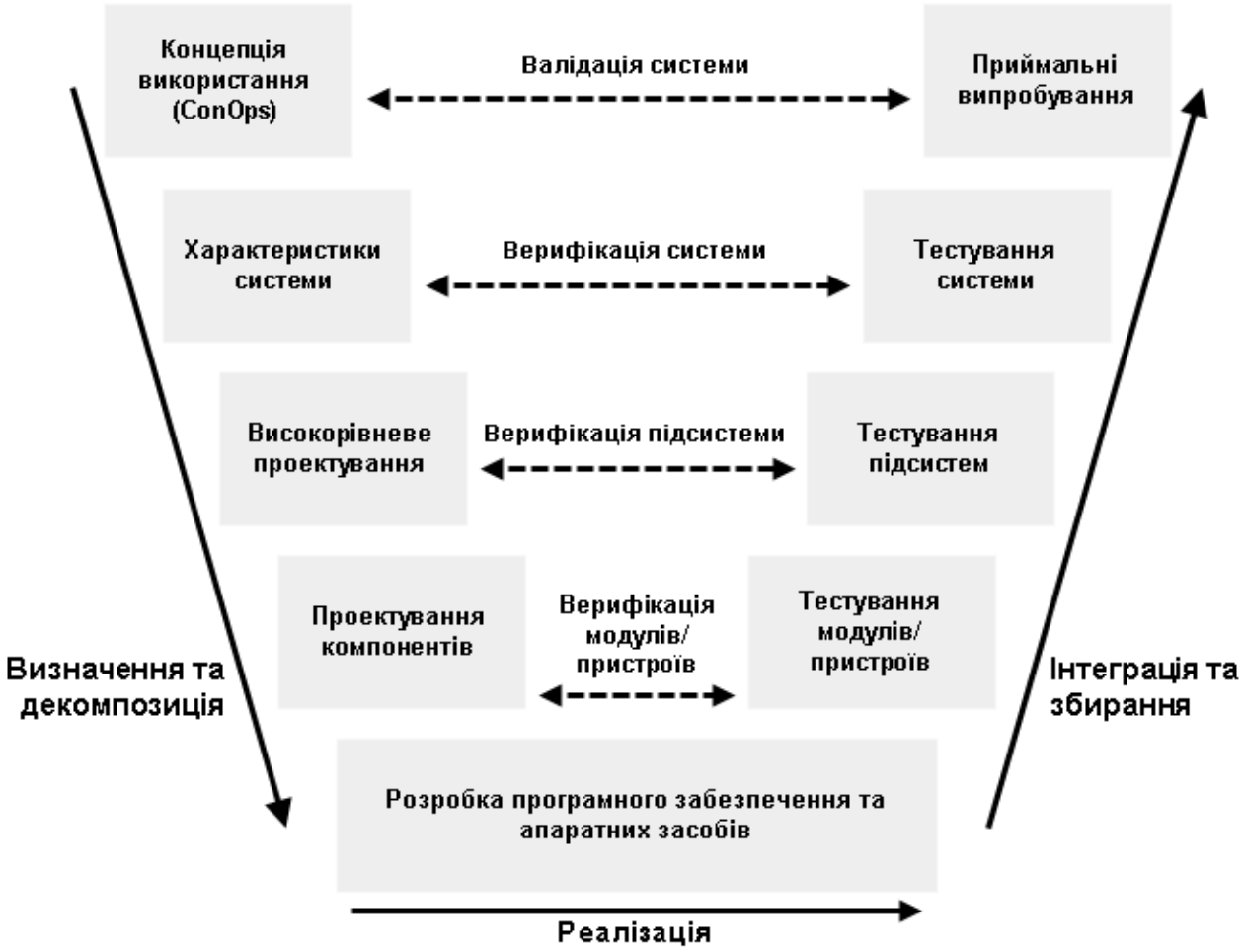

Рис. 2.4. V-модель процесу системної інженерії

- Розробка програмного забезпечення/апаратних засобів: обираються і купуються відповідні технології. Розробляється програмне забезпечення і апаратні засоби, що відповідатимуть деталізованим специфікаціям обраного проекту.
- Тестування модулів/пристроїв: відбувається тестування реалізації кожного апаратного компонента виробу, верифікується його функціональність на відповідність вимогам цього рівня.
- Тестування підсистем: відбувається інтеграція апаратних і програмних компонентів у підсистему; відбувається тестування і верифікація кожної підсистеми на відповідність високорівневим вимогам.
- Тестування системи: відбувається інтегрування підсистем і тестування всієї системи в цілому на предмет її відповідності

вимогам до системи. Проводиться верифікація, чи всі інтерфейси були коректно реалізовані і чи усіх вимог і обмежень було дотримано.

- Приймальні випробування: відбувається валідація відповідності системи поставленим вимогам і її ефективність в досягненні запланованих цілей. Протягом всього процесу системної інженерії виконується розробка і удосконалення системної документації. На кожному кроці в лівій частині V-моделі слід и вимоги, що формують і визначають наступний крок, а також план верифікації реалізації поточного рівня декомпозиції.