Тема 3 ОСНОВНІ ІНЖЕНЕРНІ ПІДХОДИ ДО СТВОРЕННЯ ПРОГРАМ

3.1. ОСНОВНІ ВІДОМОСТІ

Традиційно інженери прагнули, деякі з них, не знижуючи якості проектів, домагалися значного скорочення термінів проектування. На початку Великої Великої Вітчизняної війни начальник Центрального артилерійського конструкторського бюро В.Г. Грабін розробив та застосував методи швидкісного комплексного проектування артилерійських систем з одночасним проектуванням технологічного процесу. Впровадження цього дозволило скоротити терміни проектування, виробництва та випробувань артилерійських знарядь з 30 міс (1939) до 2 — 2,5 міс (1943), збільшити їх випуск, зменшити вартість, спростити експлуатацію.
Інженерний технологічний підхід[20] визначається специфікою комбінації стадій розробки, етапів та видів робіт, орієнтованої на різні класи програмного забезпечення та особливості колективу розробників.
Основні групи інженерних технологічних підходів та підходи для кожної з них такі:
Підходи зі слабкою формалізацією не використовують явних технологій і їх можна застосовувати лише для дуже маленьких проектів, які, як правило, завершуються створенням демонстраційного прототипу. До таких підходів відносять звані ранні технологічні підходи, наприклад підхід «кодування і виправлення».
Суворі (класичні, жорсткі, передбачувані) підходи рекомендується застосовувати для середніх, великомасштабних та гігантських проектів із фіксованим обсягом робіт. Однією з основних вимог до таких проектів є передбачуваність.
Гнучкі (адаптивні, легкі) підходи рекомендується застосовувати для невеликих або середніх проектів у разі неясних або змінних вимог до системи. При цьому команда розробників має бути відповідальною та кваліфікованою, а замовники повинні брати участь у розробці.
Класифікація технологічних підходів до створення програм:
Підходи зі слабкою формалізацією
Підхід «кодування та виправлення»
Суворі підходи
Каскадні технологічні підходи:
-класичний каскадний;
- Каскадно-поворотний;
- Каскадно-ітераційний;
— каскадний підхід з видами робіт, що перекриваються;
- Каскадний підхід з підвидами робіт;
- Спіральна модель.
Каркасні технологічні підходи:
-раціональний уніфікований підхід до видів робіт.

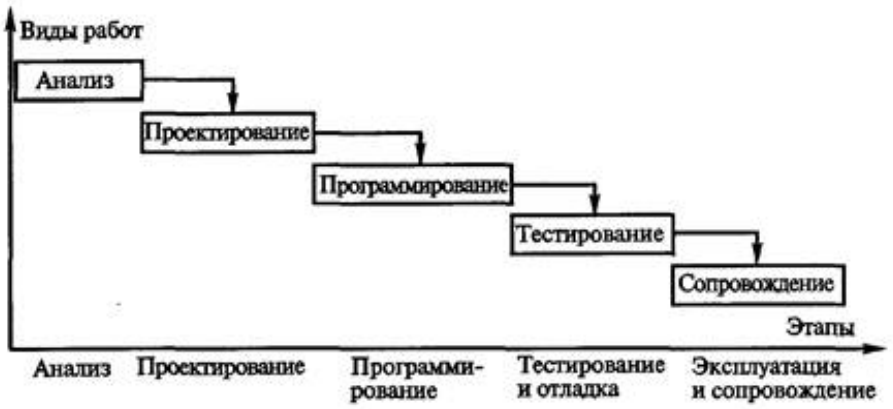
Генетичні технологічні підходи:
-синтезуюче програмування;
— складальне (розширюване) програмування;
- Конкретизуюче програмування.
Підходи на основі формальних перетворень:
-технологія стерильного цеху;
- Формальні генетичні підходи.
Гнучкі підходи
Ранні підходи швидкої розробки:
-еволюційне прототипування;
- Ітеративна розробка;
- Постадійна розробка.
Адаптивні технологічні підходи:
-екстремальне програмування;
- Адаптивна розробка;
Підходи дослідницького програмування:
-комп'ютерний дарвінізм.

3.2. РАННІ ТЕХНОЛОГІЧНІ ПІДХОДИ

Ранні технологічні підходи не використовують явних технологій, тому їх застосовують лише для дуже маленьких проектів, які зазвичай завершуються створенням демонстраційного прототипу. Як приклад підходу, не використовує формалізації, у розділі розглянуто підхід «кодування і виправлення».
Підхід «кодування та виправлення» (code and fix) спрощено може бути описаний в такий спосіб. Розробник починає кодування системи з самого першого дня, не займаючись серйозним проектуванням. Усі помилки виявляються зазвичай до кінця кодування і вимагають виправлення через повторне кодування.
Фактично кожен із програмістів так чи інакше застосовував цей підхід. При використанні цього підходу витрачається час лише на кодування та замовнику легко демонструвати прогрес у розробці рядків коду. Цей підхід може бути рекомендований для використання у двох випадках:
• для дуже маленьких проектів, які мають завершитись розробкою демонстраційного прототипу;
• для підтвердження деякої програмної концепції.

3.3. КАСКАДНІ ТЕХНОЛОГІЧНІ ПІДХОДИ

Каскадні технологічні підходи задають деяку послідовність виконання видів робіт, що зазвичай зображується у вигляді каскаду. Іноді їх називають підходами з урахуванням моделі водоспаду.
Класичний каскадний підхід(від англ. pure waterfall - чистий водоспад) вважається "дідусем" технологічних підходів до ведення життєвого циклу. Його можна як відправну точку для величезної кількості інших підходів. Сформувався класичний каскадний підхід у період з 1970 по 1985 р.
Специфіка «чистого» каскадного підходу така, що перехід до наступного виду робіт здійснюється лише після завершення роботи з поточним видом роботи (рис. 3.1). Повернення до вже пройдених видів робіт не передбачено.



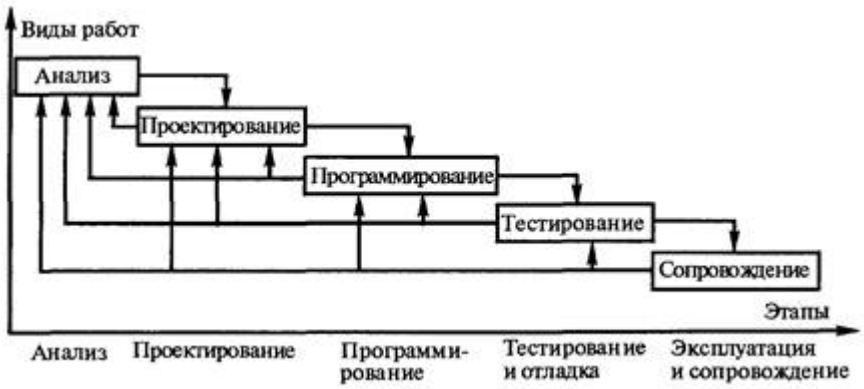
*Мал. 3.1.*Класичний каскадний підхід

Даний підхід може бути рекомендований до застосування в тих проектах, де на початку всі вимоги можуть бути сформульовані точно і повно, наприклад, в завданнях обчислювального характеру. Досить легко за такого технологічного підходу вести планування робіт і формування бюджету. Основним недоліком класичного каскадного підходу є відсутність гнучкості.

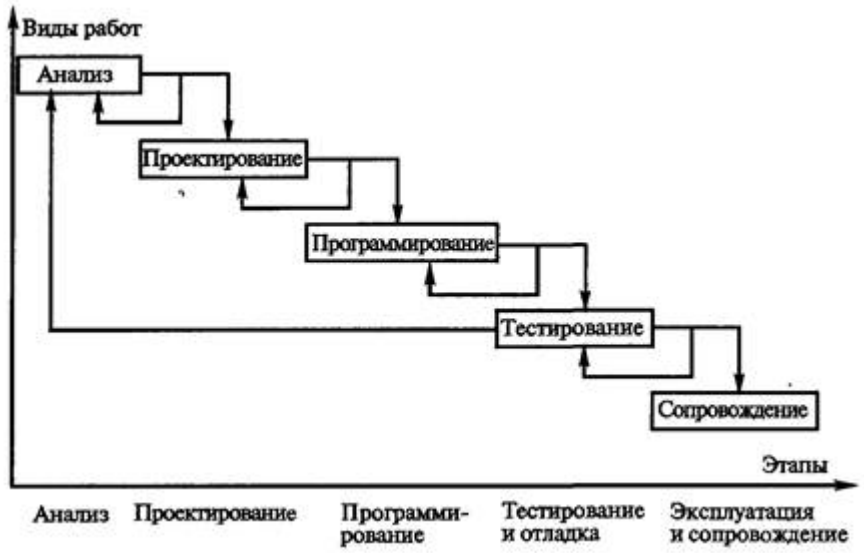
Каскадно-поворотний підхід долає нестачу класичного підходу завдяки можливості повернення до попередніх стадій та перегляду або уточнення раніше прийнятих рішень (рис. 3.2). Каскадно-поворотний підхід відбиває ітераційний характер розробки програмного забезпечення й у значною мірою реальний процес створення програмного забезпечення, зокрема і суттєве запізнення з досягненням результату. На затримку істотно впливають коригування при поверненнях.

Каскадно-ітераційний підхід передбачає послідовні ітерації кожного виду робіт доти, доки не буде досягнуто бажаного результату (рис. 3.3). Кожна ітерація є завершеним етапом і її результатом буде деякий конкретний результат. Можливо, цей результат буде проміжним, який не реалізує всю очікувану функціональність.

Каскадний підхід з видами робіт, що перекриваються.(англ. waterfall with overlapping), так само як і класичний каскадний підхід передбачає проведення робіт окремими групами розробників, але ці групи не змінюють спеціалізацію від розробки до розробки, що дозволяє розпаралелити роботи і певною мірою скоротити обсяг документації, що передається (рис. 3.4). .



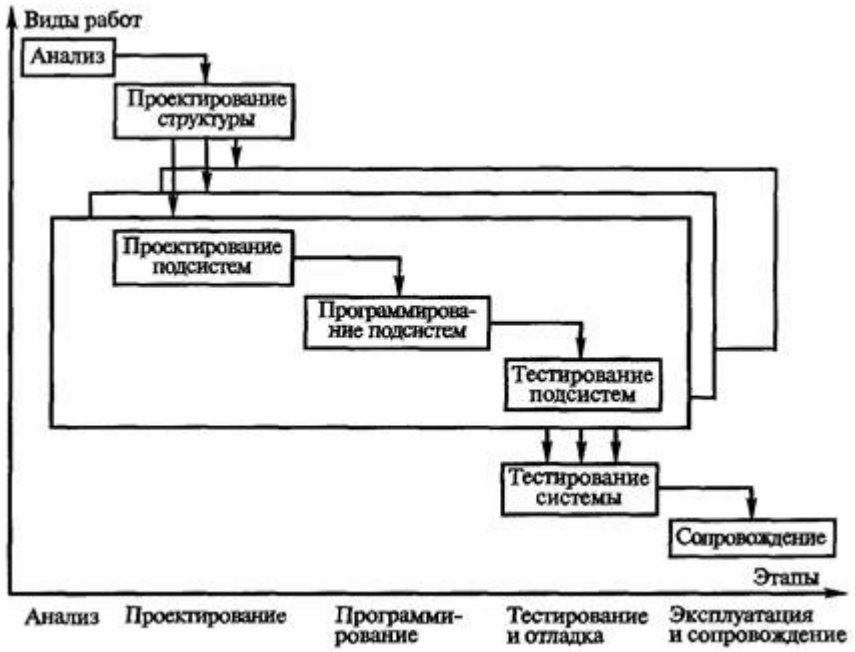
*Мал. 3.2.*Каскадно-поворотний технологічний підхід



*Мал. 3.3.*Каскадно-ітераційний технологічний підхід

*Мал. 3.4.*Каскадний підхід з видами робіт, що перекриваються.

Каскадний підхід із підвидами робіт(англ. waterfall with subprocesses) дуже близький підходу з видами робіт, що перекриваються. Особливість його в тому, що з точки зору структури проект досить часто може бути розділений на підпроекти, які можуть розроблятися індивідуально (рис. 3.5). У цьому підході потрібна додаткова фаза тестування підсистем до об'єднання в єдину систему. Слід особливу увагу звертати на грамотне розподілення проекту на підпроекти, яке має врахувати всі можливі залежності між підсистемами.



*Мал. 3.5.*Каскадний підхід із підвидами робіт

*Мал. 3.6.*Спіральна модель

Спіральна модель*(Spiral model)*була запропонована Баррі Боемом (Barry Boehm) у середині 80-х років XX ст. з метою скоротити можливий ризик розробки. Фактично це була перша реакція на старіння каскадної моделі. Спіральна модель використовує поняття прототипу - програми, що реалізує часткову функціональність програмного продукту, що створюється. Створення прототипів здійснюється у кілька витків спіралі, кожен із яких складається з «аналізу ризику», «деякого виду робіт» і «верифікації» (рис. 3.6). Звернення до кожного виду роботи випереджає «аналіз ризику», причому якщо ризик перевищення термінів та вартості проекту виявляється суттєвим, то розробка закінчується. Це дозволяє запобігти більшим грошовим втратам у майбутньому.
Особливість спіральної моделі – у розробці ітераціями. Причому кожен наступний ітераційний прототип матиме більшу функціональність.

3.4. КАРКАСНІ ТЕХНОЛОГІЧНІ ПІДХОДИ

Каркасні підходи є каркасом для видів робіт і включають їх величезну кількість.

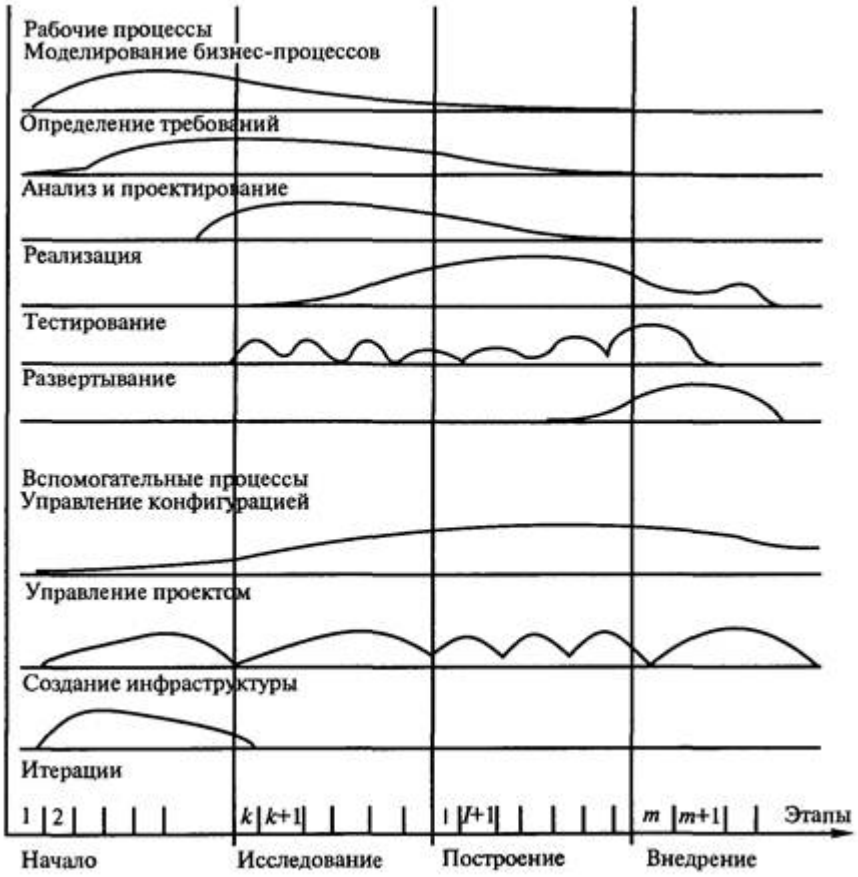
Раціональний уніфікований підхід до виконання робіт

(Rational unified process-RUP), викладений докладно в десятому розділі даного підручника, увібрав у себе найкраще з технологічних підходів каскадної групи. Вагомою перевагою цього підходу є створення інструментарію його автоматизованої підтримки — програмного продукту Rational Rose фірми «Rational Software Corpation».

З використанням підходу виділяють чотири етапи: початок, дослідження, побудова, використання. У період проходження цих етапів виконуються види робіт (наприклад, аналіз та проектування), які до того ж передбачають ітеративність їх виконання (рис. 3.7).

Основні особливості цього підходу:

• ітеративність із властивою їй гнучкістю;
• контроль якості з можливістю виявлення та усунення ризиків на ранніх етапах;
• перевага надається моделям, а не паперовим документам;
• основна увага приділяється ранньому визначенню архітектури;
• можливість конфігурування, налаштування та масштабування.



*Мал. 3.7.*Раціональний уніфікований підхід до видів робіт

3.5. ГЕНЕТИЧНІ ТЕХНОЛОГІЧНІ ПІДХОДИ

Термін «генетичний» у назві цієї групи підходів пов'язаний із походженням програми та дисципліною її створення.

Синтезуюче програмування передбачає синтез програми з її специфікації. На відміну від програми, написаної алгоритмічною мовою і призначеної для виконання на обчислювальній машині після трансляції у код, що використовується, документ мовою специфікацій є лише базисом для подальшої реалізації. Для отримання цієї реалізації необхідно вирішити такі основні завдання:

-визначити деталі, які не можна виразити за допомогою мови специфікації, але які необхідні для отримання коду, що виконується;

- Вибрати мову реалізації та апаратно-програмну платформу для реалізації;
- зафіксувати відображення понять мови специфікацій на мову реалізації та апаратно-програмну платформу;
- Здійснити трансформацію подання (зі специфікації у виконувану програму мовою реалізації);
— налагодити та протестувати програму, що виконується.

Автоматична генерація програм специфікацій можлива для багатьох мов специфікацій, особливо для SDL, ASN.1, LOTOS, Estelle, UML (Rational Rose).

Складальне (розширюване) програмування передбачає, що програма збирається шляхом повторного використання відомих фрагментів (рис. 3.8).

Складання може здійснюватися вручну або може бути задана деякою мовою складання, або витягнута напівавтоматичним чином зі специфікації завдання. Цейтін у 1990 р. виклав основні напрями для створення техніки складального програмування:
- Вироблення стилю програмування, що відповідає прийнятим принципам модульності;
- Підвищення ефективності між модульних інтерфейсів; важливість апаратної підтримки модульності;
- Ведення великої бази програмних модулів; вирішення проблеми ідентифікації модулів та перевірки придатності щодо опису інтерфейсу. (Модулі мають стати «програмними цеглинами», з яких будується програма.)

*Мал. 3.8.*Складальне програмування

Складальне програмування тісно пов'язане з методом повторного використання коду, причому як вихідного, так і бінарного. Виділяють кілька різновидів технологічних підходів складального програмування, які значною мірою визначаються базисною методологією.

1. Модульне складальне програмування - історично перший підхід, що базувався на процедурах та функціях методології структурного програмування.
2. Об'єктно-орієнтоване складальне програмування базується на методології об'єктно-орієнтованого програмування та передбачає поширення бібліотек класів у вигляді вихідного коду або упаковку класів у бібліотеку, що динамічно компонується.
3. Компонентне складальне програмування передбачає поширення класів у бінарному вигляді та надання доступу до методів класу через суворо певні інтерфейси, що дозволяє зняти проблему несумісності компіляторів і забезпечує зміну версій класів без перекомпіляції додатків, що їх використовують. Існують конкретні технологічні підходи, що підтримують компонентне програмування - COM (DCOM, COM+), CORBA, .Net. (Див. 6.6).
4. Аспектно-орієнтоване складальне програмування, у якому концепція компонента доповнюється концепцією аспекту — варіанти реалізації критичних щодо ефективності процедур. Аспектно-орієнтоване складальне програмування полягає у складанні повнофункціональних додатків з багатоаспектних компонентів, що інкапсулюють різні варіанти реалізації.

Конкретизуюче програмування передбачає, що окремі, спеціальні програми витягуються з універсальної програми.

Найбільш відома технологія конкретизуючого програмування - це підхід із застосуванням патернів проектування (див. 8.6).

Додатково до патернів існують каркаси (framework) — набори взаємодіючих класів, що становлять повторний дизайн для конкретного класу програм. Каркас диктує певну архітектуру програми, у ньому акумульовані проектні рішення, загальні для проектної галузі. Наприклад, є каркаси, які використовуються для розробки компіляторів.

3.6. ПІДХОДИ НА ОСНОВІ ФОРМАЛЬНИХ ПЕРЕТВОРЕНЬ

Ця група підходів містить максимально формальні вимоги щодо виду робіт створення програмного забезпечення.

Технологія стерильного цеху. Основні ідеї технології стерильного цеху (cleanroom process model) було запропоновано Харланом Міллзом у середині 80-х XX в. Технологія складається з наступних частин (рис. 3.9):

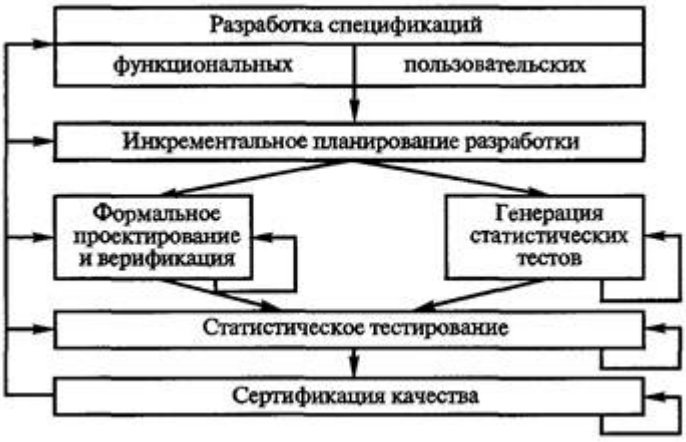
• розробка функціональних та користувальницьких специфікацій;
• інкрементальне планування розробки;
• формальна верифікація;
• статистичне тестування.

Процес проектування пов'язаний з поданням програми як функції у вигляді так званих «ящиків»:

• чорної скриньки з фіксованими аргументами (стимулами) та результатами (відповідями);
• ящика із станом, у якому виділяється внутрішній стан;
• прозорої (білої) скриньки, що представляє реалізацію у вигляді сукупності функцій при покроковому уточненні.

Використання ящиків визначають такі три принципи:

- всі визначені при проектуванні дані приховані (інкапсульовані) у ящиках;
— всі види робіт визначені як ящики, що використовують, послідовно або паралельно;
— кожну скриньку посідає певне місце у системній ієрархії.



*Мал. 3.9.*Технологія стерильного цеху

Чорний ящик являє собою точну специфікацію зовнішнього, видимого з погляду поведінки користувача. Скринька отримує стимули від користувача і видає відповідь.

Прозорий ящик отримуємо із ящика зі станами, визначаючи процедуру, яка виконує необхідне перетворення. Таким чином, прозора скринька - це просто програма, що реалізує відповідну скриньку зі станом.

Однак у цій технології відсутня такий вид робіт, як налагодження. Його замінює процес формальної верифікації. Для кожної структури, що управляє, перевіряється відповідна умова коректності.

Технологія стерильного цеху передбачає бригадну роботу, т. е. проектування, уточнення, інспекцію та підготовку текстів ведуть різні люди.

Формальні генетичні підходи Склалися методи програмування, які мають властивість доказовості. Три такі методи відповідають вже дослідженим генетичним підходам, але з урахуванням формальних математичних специфікацій.

Формальне синтезуюче програмування використовує математичну специфікацію - сукупність логічних формул. Існують два різновиди синтезуючого програмування: логічне, в якому програма витягується як конструктивний доказ зі специфікації, що розуміється як теореми, та трансформаційний, в якому специфікація розглядається як рівняння щодо програми та символічними перетвореннями перетворюється на програму.

Формальне складальне програмування використовує специфікацію як композицію відомих фрагментів. *Формальне конкретизуюче програмування* використовує такі підходи, як змішані обчислення та конкретизацію щодо анотацій.

3.7. РАННІ ПІДХОДИ ШВИДКОЇ РОЗРОБКИ

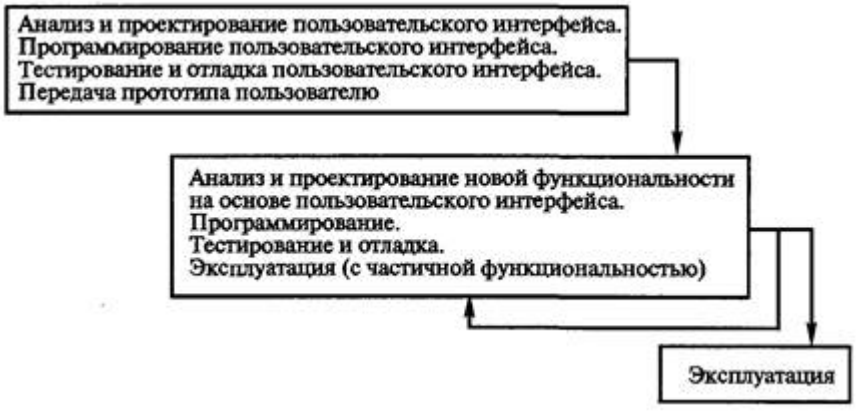
Розвитком та одночасно альтернативою каскадних підходів є група підходів швидкої розробки. Всі ці підходи поєднують такі основні риси:
• ітераційну розробку прототипу;
• тісна взаємодія із замовником.

Еволюційне прототипування. Перший прототип при еволюційному прототипуванні (evolutionary prototyping) зазвичай включає створення розвиненого інтерфейсу користувача, який може бути відразу ж продемонстрований замовнику для отримання від нього відгуків і можливих коректив. Основна початкова увага приділяється стороні системи, зверненої до користувача. Далі, доки користувач не визнає програмний продукт закінченим, до нього вноситься необхідна функціональність (рис. 3.10). Еволюційне прототипування розумно застосовувати в тих випадках, коли замовник не може чітко сформулювати свої вимоги до програмного продукту на початкових етапах розробки, або замовник знає, що вимоги можуть кардинально змінитись.

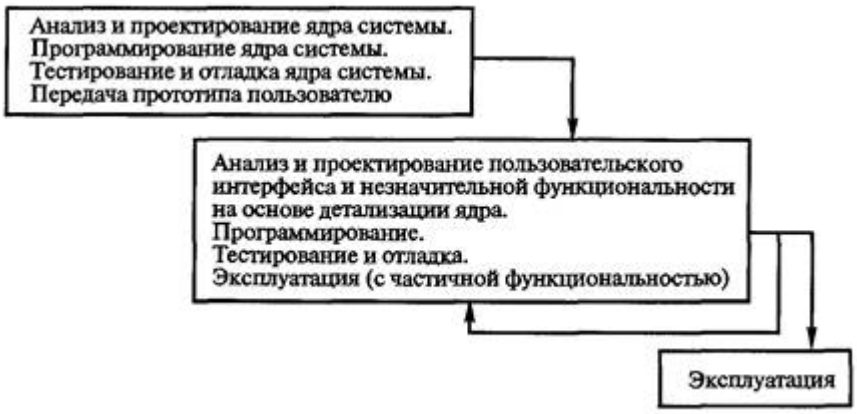
Істотним недоліком цього підходу є неможливість визначити тривалість та вартість проекту. Неочевидною є кількість ітерацій, після яких користувач визнає програмний продукт закінченим.

Ітеративна технологія. Перший прототип ітеративної розробки (iterative delivery) вже повинен містити завершене ядро системи. Таким чином, у ньому вже зосереджена більшість функціональності. Чергові ітерації повинні допомогти користувачеві визначитися з доведенням інтерфейсу користувача та генерованих систем звітів та інших вихідних даних (рис. 3.11).

Допускається додавання незначної функціональності, що зазвичай не торкається ядра системи. Відмінність між двома дослідженими методами швидкої розробки полягає в тому, що ітеративна розробка починається зі створення ядра системи і далі деталізує його, а еволюційне прототипування орієнтоване на початкову розробку інтерфейсу користувача і додавання функціональності на його основі.



*Мал. 3.10.*Еволюційне прототипування



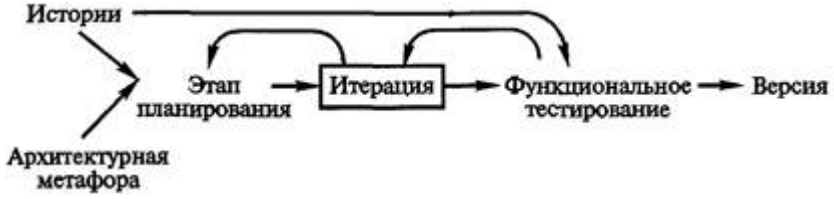
*Мал. 3.11.*Ітеративна розробка

Постадійна технологія (staged delivery) призначена усунути нестачу двох попередніх підходів - неможливість визначення термінів завершення проекту. Починаючи розробку, розробник досить добре знає, що буде являти собою створюваний програмний продукт. Основне завдання постадійної розробки – надати замовнику працюючу систему якомога раніше. Далі замовник зможе додавати нову функціональність та отримувати чергову версію системи. Однак кожна з версій, що отримуються після завершення стадій, є чинною.
Цей підхід вимагає ретельного та серйозного тестування чергової системи наприкінці кожної стадії перед передачею її користувачеві.

3.8. АДАПТИВНІ ТЕХНОЛОГІЧНІ ПІДХОДИ

Адаптивні технологічні підходи були задумані як підходи, які підтримують зміни. Вони лише виграють від змін, навіть коли зміни відбуваються у них самих. Дані підходи спрямовані на людини, а чи не на процес. Під час роботи в них необхідно враховувати природні якості людської натури, а не діяти їм наперекір (http://www.martinfowler.com).
Екстремальне програмування. Найбільш концентровано ідеї швидкої розробки програм виявились у підході екстремального програмування (extreme programming) (XP) (http://www.extremeprogramming.org). Дві основні риси, властиві швидким розробкам, є основними і в цьому підході. Методи, об'єднані у цьому підході, є принципово новими. Проте саме їхнє раціональне об'єднання та сукупне використання дають суттєві результати та успішно виконані проекти. Найбільшу користь підхід екстремального програмування може принести розробці невеликих систем, вимоги до яких чітко не визначені і можуть змінитися.
Як відбувається традиційний процес розробки програмного забезпечення? Організується група аналітиків, яка прикріплюється до проекту. Група аналітиків протягом кількох годин на тиждень зустрічається з ймовірними користувачами, після чого вони випускають документацію на проект і приступають до його обговорення.
Використовуючи надану їм специфікацію, програмісти після декількох місяців випускають програмний продукт, який більш-менш відповідає тому, що від нього очікують. Найчастіше до завершення проекту ситуація може змінитися і користувачі захочуть внести зміни чи додавання, які здійснитися на даний момент не можуть. Тому програмісти, провівши тестування, здають програмний проект замовнику у вигляді, як він був замовлений. Замовник змушений розпочати фінансування розробки нової версії програми.
Екстремальне програмування дозволяє залучити кінцевих користувачів для тестування вже на ранніх етапах проектування та розробки системи. При цьому замовник звертається до розробників із проханням виготовити програмну систему. Протягом усієї роботи над проектом потрібна присутність представника замовника. Проект поділяється на три етапи:

— етап планування реалізації: замовники пишуть сценарії роботи системи на основі списку історій — можливих застосувань системи, програмісти адаптують їх до розробки, після чого замовники обирають першочергову з написаних сценаріїв;
— ітераційний етап: замовники пишуть тести та відповідають на запитання розробників, доки останні програмують;
- Етап випуску версії: розробники встановлюють систему, замовники приймають роботу.
Замовник в екстремальному програмуванні має багато можливостей вплинути на напрям роботи команди, оскільки ітерації проекту видають кожні два тижні програмне забезпечення, яке можна використовувати та тестувати. Беручи таку активну участь у розробці, замовник може бути впевненим у актуальності інформації, яка використовується в роботі системи.
В екстремальному програмуванні існує фундаментальний поділ ролей замовників та розробників, які працюють в одній команді, але мають різні права у прийнятті рішень. Замовник вирішує «що потрібно отримати», тоді як розробник вирішує «скільки це коштуватиме» і «скільки часу це займе».
Практика екстремального програмування дозволяє розділити відповідальність між замовником та розробником. Поділ робочої сили дозволяє команді виконувати роботу точно вчасно, не втративши при цьому актуальність системи. Наприклад, якщо замовник хоче, щоб програма генерувала нові звіти на цьому тижні, розробники готові це надати. Але вони повинні повідомляти про можливі технічні ризики (якщо такі є) і вартість робіт з внесення змін.
Як вирішуються конфлікти? Що відбувається, коли замовник хоче отримати продукт до певної дати, але розробнику потрібно на його виготовлення трохи більше часу? Екстремальне програмування пропонує кілька варіантів рішення: замовник приймає систему з меншими функціональними можливостями, замовник приймає пізнішу дату, замовник приймає рішення витратити гроші або час на розробку альтернативного варіанту або замовник може знайти іншу команду розробників.
Підхід починається з аналізу призначення системи та визначення першочергової функціональності. У результаті складається список історій – можливих застосувань системи. Кожна історія має бути орієнтована на певні завдання бізнесу, які можна оцінити за допомогою кількісних показників. Зрештою, цінність історії визначається матеріальними та тимчасовими витратами на її розробку командою розробників.
Замовник вибирає історії для чергової ітерації, ґрунтуючись на їх значущості для проекту та цінності. Для першої версії системи замовник визначає невелику кількість логічно пов'язаних найважливіших історій. Для кожної наступної версії вибираються найважливіші історії з числа історій, що залишилися (рис. 3.12).

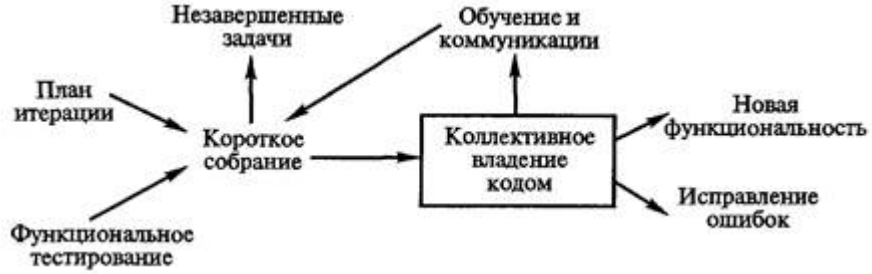


*Мал. 3.12.*Робота над проектом на основі екстремального програмування

Одним із суттєвих методів даного підходу є функціональне тестування. Існують дві особливості процесу тестування:
• програмісти пишуть тести для тестування програми;
• ці випробування пишуться до початку кодування.
Для будь-якого автономного модуля (класу, процедури, Unit-модуля) програмісти пишуть окремий модульний тест, який має тестувати всі основні варіанти використання цього модуля та зберігатися разом із ним. Результати прогонів тестів мають бути лаконічними, наприклад, «OK! (10 tests)». Головне – тест повинен писатися до написання самого модуля! Таке тестування називають випереджальним.
Внесення змін до кожної ітерації проекту (рефакторинг) завжди супроводжується прогоном всіх тестів, щоб гарантувати стабільну роботу системи. Впевненість у нормальній роботі як кожного окремого тесту, так і всіх тестів комплексного тесту надає розробникам впевненості у нормальній роботі чергової версії системи на кожній ітерації проекту.

Мета кожної ітерації (рис. 3.13) – включити до версії кілька нових історій. На зборах із планування ітерації визначається, які саме історії та яким чином вони будуть реалізовані командою розробників. Колективне володіння кодом у процесі розробки (рис. 3.14) означає можливість кожному програміста у час удосконалити будь-яку частину коду у системі, якщо він вважатиме це за необхідне. Програміст перебирає відповідальність за реалізацію певних завдань. У разі виникнення питань щодо розроблюваного завдання можуть бути проведені короткі (15-хвилинні) збори в присутності замовника.

Мал. 3.13. Робота на ітерації екстремального програмування



*Мал. 3.14.*Колективне володіння кодом при екстремальному програмуванні

Щоб виконати завдання, відповідальний за неї програміст повинен знайти партнера (рис. 3.15). Остаточний код завжди пишеться двома програмістами на одній робочій станції.
Екстремальне програмування приділяє значну увагу організації офісного простору, наголошуючи на суттєвому впливі навколишніх умов на роботу програмістів.
Адаптивна технологія. В основу підходу адаптивної розробки (Adaptive Software Development — ASD) покладено три нелінійні етапи, що перекривають один одного — обмірковування, співробітництво та навчання. Автор цього підходу Джим Хайсміт (Jim Highsmith) звертає особливу увагу на використання ідей в галузі складних адаптивних систем.
Результати планування (яке саме тут є парадоксальним) у адаптивній розробці завжди будуть непередбачуваними. При звичайному плануванні, відхилення від плану є помилкою, яку виправляють. При цьому відхилення ведуть до правильних рішень.

*Мал. 3.15.*Робота над кодом парою програмістів при екстремальному програмуванні

Програмісти повинні активно співпрацювати між собою для подолання невизначеності у підході адаптивної розробки. Керівники проектів повинні забезпечити хороші комунікації між програмістами, завдяки чому програмісти самі знаходять пояснення на питання, що виникають, а не чекають їх від керівників.

Навчання є постійною та важливою характеристикою підходу. І програмісти, і замовники у процесі роботи мають переглядати власні зобов'язання та плани. Підсумки кожного циклу розробки використовуються під час підготовки наступного.

3.9. ПІДХОДИ ДОСЛІДНОГО ПРОГРАМУВАННЯ

Дослідницьке програмування має такі особливості (http://www.osp.ru/pcworld/2001/01/062.htm):
— розробник ясно уявляє напрямок пошуку, але не знає заздалегідь, як далеко він зможе просунутися до мети;
- немає можливості передбачати обсяг ресурсів для досягнення того чи іншого результату;
- розробка не піддається детальному плануванню, вона ведеться методом спроб та помилок;
— робота пов'язана з конкретними виконавцями та відображає їх особисті якості.
В основі дослідницького програмування більшою мірою, ніж в інших підходах, лежить мистецтво.
Комп'ютерний дарвінізмНазва цього підходу було запропоновано Кеном Томпсоном (Ken Thompson). Підхід заснований на принципі висхідної розробки, коли система будується навколо ключових компонентів та програм, що створюються на ранніх стадіях проекту, а потім постійно модифікуються. Все більші блоки збираються з раніше створених дрібних блоків.
Комп'ютерний дарвінізм є методом спроб і помилок, заснований на інтенсивному тестуванні, причому на будь-якому етапі система повинна працювати, навіть якщо це мінімальна версія того, чого прагнуть розробники. Природний відбір залишить лише життєздатне.
Підхід складається з трьох основних видів робіт:
• макетування (прототипування);
• тестування;
• налагодження.
Однією з цікавих особливостей підходу є максимально можливе розпаралелювання процесів тестування та налагодження.

ВИСНОВКИ

• Традиційно інженери прагнули, а деякі з них домагалися, не знижуючи якості проектів, значного скорочення термінів проектування.
• Інженерний технологічний підхід визначається специфікою комбінації стадій та видів робіт, орієнтованої на різні класи програмного забезпечення та на особливості колективу розробників.
• Основні групи інженерних технологічних підходів: підходи зі слабкою формалізацією, суворі та гнучкі (адаптивні) підходи.
• Ранні технологічні підходи не використовують явні технології і їх зазвичай застосовують тільки для дуже маленьких проектів.
• Каскадні технологічні підходи задають деяку послідовність виконання видів робіт, що зазвичай зображується у вигляді каскаду (водоспаду).
• Каркасні підходи є каркасом для видів робіт. Одним із яскравих представників каркасного підходу є раціональний уніфікований підхід до виконання робіт, підтримуваний САПР на основі програмного продукту Rational Rose фірми Rational Software Corporation.
• Термін «генетичний» у назві групи генетичних технологічних підходів пов'язується з походженням програми та дисципліною її створення.
• Адаптивні технологічні підходи були задумані як підходи, що підтримують зміни. Вони лише виграють від змін, навіть коли зміни відбуваються у них самих. Найбільш підходом цієї групи, що бурхливо розвивається, є екстремальне програмування.
• Вибір оптимального інженерного технологічного підходу забезпечує скорочення термінів виконання проекту без зниження якості проекту.

Контрольні питання

1. За рахунок чого інженери вимагають різкого скорочення термінів виконання проектів?
2. Які відомі основні групи інженерних технологічних підходів?
3. Яка класифікаційна ознака покладена в основу поділу інженерних технологічних підходів на основні групи?
4. Які інженерні технологічні підходи розвинулися останнім часом?

5. У чому полягають переваги та недоліки каскадно-поворотного підходу порівняно з класичним каскадним підходом?
6. Який вид робіт відсутній у технології стерильного цеху?
7. У яких випадках розумно застосовувати еволюційне прототипування?
8. Які дві основні риси, властиві швидким розробкам, є базовими під час екстремального програмування?
9. У чому полягає суть адаптивної розробки?