**ЛЕКЦІЯ-4. Життєвий цикл програмного забезпечення**

1. **Життєвий цикл програмного забезпечення**

***Життєвий цикл ПЗ (ЖЦПЗ)*** – це безперервний процес, який починається з моменту прийняття рішення про необхідність створення ПЗ і закінчується в момент його повного вилучення з експлуатації.

Основним нормативним документом, який регламентує ЖЦ, є міжнародний стандарт ISO/IEC 12207. Він визначає його структуру, містить процеси, дії і задачі, які повинні бути виконані під час створення ПЗ. Структура ЖЦ за стандартом ISO/IEC 12207 базується на трьох групах процесів:

* основні процеси (придбання, постачання, розробка, експлуатація, супровід);
* допоміжні процеси (документування, управління конфігурацією, забезпечення якості, перевірка, атестація);
* організаційні процеси (управління проектами, створення інфраструктури проекту, визначення, оцінка і покращення самого життєвого циклу, навчання).

***Розробка включає в себе*** всі роботи по створенню ПЗ і його компонентів відповідно до заданих вимог, включаючи оформлення проектної і експлуатаційної документації, підготовку матеріалів, необхідних для перевірки працездатності і якості програмних продуктів, матеріалів, необхідної для організації навчання персоналу і т.д. Розробка ПЗ включає в себе, як правило, аналіз, проектування і реалізацію (програмування).

***Експлуатація включає в себе*** роботи по впровадженню компонентів ПЗ в експлуатацію, в тому числі конфігурацію бази даних, і робочих місць користувачів, забезпечення експлуатаційною документацією, проведення навчання персоналу, і безпосередньо експлуатацію, модифікацію ПЗ, підготовку пропозицій до вдосконалення, розвитку і модернізації системи.

***Управління проектом*** пов’язане з питаннями планування і організації робіт, створення колективів розробників і контролю за термінами і якістю виконуваних робіт.

**Технічне і організаційне забезпечення проекту** включає вибір методів і інструментальних засобів для реалізації проекту, визначення методів опису проміжних станів розробки, розробку методів і засобів випробувань ПЗ, навчання персоналу. Забезпечення якості проекту пов’язане з проблемами перевірки і тестування ПЗ. Перевірка – це процес визначення того, чи відповідає поточний стан розробки, досягнутий на даному етапі, вимогам цього етапу.

***Управління конфігурацією*** є одним з допоміжних процесів, який підтримує основні процеси життєвого циклу ПЗ, передусім процеси, розробки і супроводу ПЗ. При створенні проектів складних інформаційних систем, які складаються з багатьох компонентів, кожен з яких може мати версії, виникає проблема обліку їх зв’язків, створення уніфікованої структури і забезпечення розвитку всієї системи. Управління конфігурацією дозволяє організувати, систематично враховувати і контролювати внесення змін в ПЗ на всіх стадіях життєвого циклу.

Кожний процес характеризується певними задачами і методами їх вирішення, початковими даними, отриманими на попередньому етапі, і результатами. Результатами аналізу, зокрема, є функціональні моделі, інформаційні моделі і відповідні їм діаграми.

1. **Моделі ЖЦ**

**Модель ЖЦ** – структура, яка визначає послідовність виконання та взаємозв’язку процесів, дій та задач на протязі виконання проекту. Модель ЖЦ залежить від специфіки програмного продукту, а також від специфіки умов, в яких даний продукт створюється та функціонує.

**Існує два основних типи моделей ЖЦ:**

* ***прогнозовані моделі*** ЖЦ – в основі цих моделей лежить чітке планування усіх стадій процесу розробки ПЗ.
* ***адаптивні моделі ЖЦ*** (так звані гнучкі технології) – їх особливістю є сприйняття та адаптація процесів розробки під потреби замовника на ринку.

На даний час найпоширенішими ***серед прогнозованих моделей*** є:

* Каскадна модель;
* V-подібна модель;
* Інкрементна модель;
* Спіральна модель;
* Модель швидкої розробки;
* Модель еволюційного прототипування.

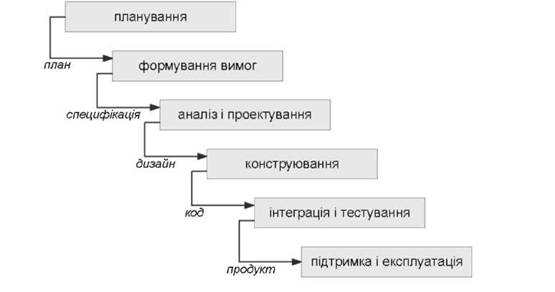
Найвідомішими ***адаптивними моделями ЖЦ*** є:

* Scrum;
* XP (Extreme Programming);
* Adaptive Software development (ASD);
* Dynamic System Development Model (DSDM);
* Feature Driven Development (FDD)

Адаптивні моделі ЖЦ включають не тільки опис структури фаз, та їх пояснення, але і рекомендації, підходи їх ефективного використання.

## Каскадна модель ЖЦ

Була розроблена Вінстоном Ройсом в 1970 році. Класична модель ЖЦПЗ застосовується як для розробки програм для індивідуального використання, так і для складних програмних виробів промислового і комерційного призначення. Дуже часто класичний ЖЦ називають каскадною або водоспадною моделлю, підкреслюючи, що розробка розглядається як послідовність етапів, причому перехід на наступний, ієрархічно нижчий етап відбувається тільки після завершення робіт на поточному етапі.



На практиці виконання робіт за такою жорсткою схемою можна реалізувати тільки для досить простих програмних продуктів, вимоги користувача до яких можна на самому початку сформулювати в повному обсязі. У загальному випадку процес розробки є ітеративним. На кожній фазі ЖЦ можуть вноситися зміни в прийняті рішення, що приводить до необхідності повернення до попередніх етапів.

Вважається, що розробка починається на системному рівні і проходить через аналіз, проектування, кодування, тестування і супровід. При цьому моделюються дії стандартного інженерного циклу.

*Планування.* У ході планування проекту визначаються обсяг проектних робіт і їх ризик, необхідні трудові затрати, формуються робочі завдання і план-графік робіт. Результатом роботи даного етапу є визначення вимог користувача.

*Аналіз вимог* відноситься до програмного елементу – ПЗ. Уточнюються і деталізуються його функції, характеристики та інтерфейс. Усі визначення документуються в специфікації аналізу. На цьому ж етапі завершується вирішення задачі планування проекту. Результатом роботи є визначення вимог до ПЗ.

Проектування полягає у створенні рішень:

* Архітектури ПЗ;
* Модульної структури ПЗ;
* Алгоритмічної структури ПЗ;
* Структури даних;
* Вхідного і вихідного інтерфейсів (вхідних і вихідних форм даних).

Вихідні дані для проектування містяться в специфікації аналізу, тобто в ході проектування виконується трансляція вимог до ПЗ в множину проектних рішень. При вирішенні задач проектування основна увага приділяється якості майбутнього програмного продукту. Результатом проектування є архітектура програмного виробу.

Кодування полягає в перекладі результатів проектування на текст мовою програмування.

Тестування – виконання програми для виявлення дефектів в функціях, логіці і формі реалізації програмного продукту.

Супровід – це внесення змін в працююче ПЗ. Мета внесення змін:

* Виправлення помилок;
* Адаптація до змін зовнішнього для ПЗ середовища;
* Удосконалення ПЗ на вимогу замовника.

Супровід ПЗ полягає в повторному застосуванні кожного з попередніх кроків (етапів) життєвого циклу до існуючої програми, але не в розробці нової програми.

Як і будь-яка інженерна система, класичний ЖЦ має переваги й недоліки. До переваг класичного ЖЦ можна віднести:

* можливість планувати в часі всі етапи проекту;
* впорядкування ходу конструювання.

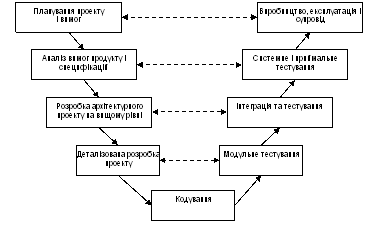
Недоліками класичного ЖЦ є:

* реальні проекти часто вимагають відхилення від стандартної послідовності кроків;
* цикл базується на точному формулюванні вихідних вимог до ПЗ;
* результати проекту доступні замовнику лише після завершення роботи.

## V-подібна модель

V-подібна модель була створена для допомоги працюючій над проектом «команді» в плануванні та забезпеченні подальшої можливості тестування системи. В цій моделі особливе значення надається дiям, спрямованим на верифiкацiю й атестацію продукту. План приймання кінцевого програмного продукту замовником розробляється на етапi планування,а планування решти системи – на фазах аналізу, розробки проекту тощо.

Процес розробки планiв тестування позначений пунктирною лiнiєю між прямокутниками V-подiбної моделi



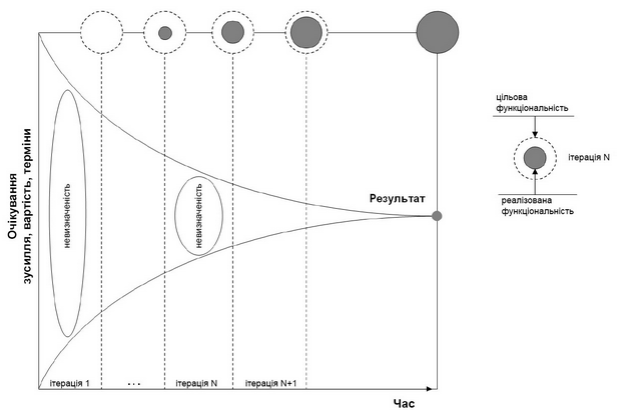
Проте, незважаючи на всi свої переваги та спрямованiсть на тестування, V-подiбна модель має послiдовну структуру: кожна наступна фаза починається лише по завершенню попередньої.

Незважаючи на те, що передбачається план тестування розробляти на ранніх етапах ЖЦПЗ, саме тестування здiйснюється вже пiсля того, як продукт створено.

Отже, помилки, виявленi на цьому етапi (пов’язанi з неправильним проектуванням) спричинять повернення на попереднi етапи життєвого циклу, а отже призведуть до значних фінансових втрат.

## Інкрементна модель

Ітеративна, що отримала також від Т. Гілба в 70-і рр. назву еволюційної моделі ( ітеративна моделлю та інкрементна модель).

Модель IID передбачає розбиття ЖЦ проекту на послідовність ітерацій, кожна з яких нагадує "міні-проект", включаючи всі процеси розробки в застосуванні до створення менших фрагментів функціональності, порівняно з проектом в цілому. Мета кожної ітерації — отримання працюючої версії програмної системи, що включає функціональність, визначену інтегрованим змістом всіх попередніх та поточної ітерації. Результат фінальної ітерації містить всю необхідну функціональність продукту. Таким чином, із завершенням кожної ітерації продукт отримує приріст — інкремент — до його можливостей, які, значить, розвиваються еволюційно. З точки зору структури ЖЦ таку модель називають ітеративною. З точки зору розвитку продукту — інкрементальною. Досвід індустрії показує, що неможливо розглядати кожен з цих поглядів ізольовано.   
  


При такому підході кожна версія повинна бути придатною для використання і забезпечувати деяку підмножину необхідних функцій. До недоліків відноситься збільшення обсягу робіт по тестуванню, оскільки з появою кожної нової версії продукту потрібне повторне підтверджуюче тестування, що додані можливості не погіршили якість програмного виробу.

Кожна лінійна послідовність виробляє працюючий інкремент ПЗ. Перший інкремент приводить до отримання базового продукту, який реалізує базові вимоги (більшість додаткових вимог залишаються нереалізованими). План наступного інкременту передбачає модифікацію базового продукту, яка забезпечить додаткові характеристики і функціональність.

Сучасна реалізація інкрементного підходу – екстремальне програмування XP. Воно орієнтоване на дуже невеликі прирости функціональності.

## Спіральна модель

Спіральна модель запропонована Баррі Боемом в 1988 році. Вона базується на кращих властивостях класичного життєвого циклу, до якого добавлено новий елемент – аналіз ризику.

Спіральна схема розробки ПЗ представляється у вигляді чотирьох квадрантів, які визначають чотири види діяльності, що виконуються послідовно:

* планування – визначення цілей, варіантів і обмежень;
* аналіз ризику – аналіз варіантів і розпізнавання ризику;
* конструювання – розробка продукту наступного рівня;
* оцінювання – оцінка замовником поточних результатів конструювання.

На першому витку спіралі визначаються початкові цілі, варіанти і обмеження, розпізнаються і аналізуються ризики. Якщо аналіз ризику показує невизначеність вимог, на допомогу розробнику і замовнику приходить макетування, яке використовується в квадранті конструювання. Для подальшого визначення проблемних і уточнених вимог можна застосувати моделювання. Замовник оцінює конструкторську роботу і вносить пропозиції по модифікації. Наступна фаза планування і аналізу ризику базується на пропозиціях замовника. В кожному циклі спіралі результати аналізу ризику формуються у вигляді „продовжувати, не продовжувати”. Якщо ризик дуже великий, проект може бути зупинений.

У більшості випадків рух по спіралі продовжується, з кожним кроком просовуючи розробника до більш загальної моделі системи. На кожному циклі спіралі потрібне конструювання, яке може бути реалізоване класичним ЖЦ, або макетуванням. Кількість дій при конструюванні зростає в міру просовування від центру спіралі.

Перевагами спіральної моделі є:

* найбільш реально (у вигляді еволюції) відображає розробку ПЗ;
* дозволяє явно врахувати ризик на кожному витку еволюції розробки;
* включає крок системного підходу в ітераційну структуру розробки;
* використовує моделювання для зменшення ризику і удосконалення ПЗ.

Недоліки спіральної моделі:

* новизна (відсутня достатня статистика ефективності моделі);
* підвищені вимоги до замовника;
* важкість контролю і управління часом розробки.

## Компонентно-орієнтована модель

Компонентно-орієнтована модель є розвитком спіральної моделі і також базується на еволюційній стратегії конструювання. В цій моделі конкретизується зміст квадранта конструювання – він відображає той факт, що в сучасних умовах нова розробка повинна базуватися на повторному використанні існуючих програмних компонентів.

Програмні компоненти, які створені в реальних програмних проектах, зберігаються в бібліотеках. В новому програмному проекті, виходячи з вимог замовника, виявляються кандидати в компоненти. Далі перевіряється наявність цих кандидатів в бібліотеці. Якщо вони знайдені, то компоненти виймаються з бібліотеки і використовуються повторно. В іншому випадку створюються нові компоненти, вони використовуються в проекті і включаються в бібліотеку.

Переваги компонентно-орієнтованої моделі:

* зменшує на 30% час розробки програмного продукту;
* зменшує вартість програмної розробки на 70%;
* збільшує в півтора рази продуктивність розробки.

## Модель швидкої розробки програм

Модель швидкої розробки програм (RAD – Rapid Application Development) – інший приклад застосування інкрементної стратегії конструювання. RAD-модель забезпечує дуже короткий цикл розробки. RAD – високошвидкісна адаптація лінійної послідовної моделі, в якій швидка розробка досягається за рахунок використання компонентно-орієнтованого конструювання. Якщо вимоги повністю визначені, а проектна галузь обмежена, RAD-процес дозволяє групі створити повністю функціональну систему за дуже короткий термін (60-90 днів). RAD-підхід орієнтований на розробку інформаційних систем і виділяє наступні етапи:

* Бізнес-моделювання. Моделюється інформаційний потік між бізнес-функціями. Шукається відповіді на питання: Яка інформація керує бізнес-процесом? Яка інформація генеруються? Хто її генерує? Де вона використовується? Хто її обробляє?
* Моделювання даних. Інформаційний потік, визначений на етапі попередньому етапі, відображається в набір об’єктів даних, які потрібні для підтримки бізнесу. Ідентифікуються характеристики (властивості, атрибути) кожного об’єкту, визначаються відношення між ними.
* Моделювання обробки. Визначаються перетворення об’єктів даних, які забезпечують реалізацію бізнес-функцій. Створюються описи обробки для додавання, модифікації, знищення або знаходження (виправлення) об’єктів даних.
* Генерація програми. Передбачається використання методів, які орієнтовані на мови програмування 4-го покоління. Замість створення ПЗ за допомогою мов програмування 3-го покоління, RAD-процес працює з повторно використовуваними програмними компонентами, або створює повторно використовувані компоненти. Для забезпечення конструювання використовуються утиліти автоматизації.
* Тестування і об’єднання. Оскільки застосовуються повторно використовувані компоненти, більшість програмних елементів вже налагоджені. Це зменшує час тестування (хоча всі нові елементи повинні бути налагоджені).

На відміну від традиційного підходу, при якому використовувалися специфічні засоби створення прототипів, які не призначені для побудови реальних програм, прототипи викидалися після того, як була виконана задача усунення неясностей в проекті, в підході RAD кожний прототип розвивається в частину майбутньої системи.

На фазі побудови виконується безпосередньо швидка розробка ПЗ – розробники проводять інтерактивну побудову реальної системи на основі отриманих в попередній фазі моделей, а також вимог нефункціонального характеру. Програмний код частково формується за допомогою автоматичних генераторів. Замовники на цій фазі оцінюють отримані результати і вносять корективи. Тестування системи здійснюється безпосередньо в процесі розробки.

Після закінчення робіт кожної команди розробників проводиться поступова інтеграція даної частини системи з іншими, формується повний програмний код, виконується тестування спільної роботи даної частини програми з іншими, а потім тестування системи загалом. Результатом фази є готова система, яка задовольняє всі узгоджені вимоги.

На фазі впровадження проводиться навчання користувачів, організаційні зміни і паралельно з впровадженням нової системи здійснюється робота з існуючою системою.

Використання RAD має свої недоліки і обмеження.

* для великих проектів в RAD потрібні великі людські ресурси;
* RAD застосовується тільки для таких програм, які можна розділити на окремі модулі і в який продуктивність не є критичною величиною;
* RAD не застосовують в умовах великих технічних ризиків.

Методологія RAD непридатна для побудови складних розрахункових програм, ОС, або програм реального часу. Не підходять для розробки за методологією RAD програми, в яких відсутня яскраво виражений інтерфейс, який наочно визначає логіку роботи системи і програм, від яких залежить безпека людей, оскільки інтерактивний підхід передбачає, що перші декілька версій не будуть повністю робочими.

## Модель еволюційного прототипування ( Макетування)

Досить часто користувач-замовник формулює тільки загальні цілі програмного продукту і не визначає усі вимоги. З іншого боку, розробник може сумніватися в сумісності продукту з ОС, формі діалогу з користувачем, або в ефективності реалізованого алгоритму. В цьому випадку розробка прототипу програмного продукту часто виявляється найкращою моделлю процесу розробки. В прототипі прагнуть реалізувати вирішення основних проблемних питань, поставлених замовником, і продемонструвати йому їх реалізацію.

Головна мета створення прототипу – уточнення вимог користувача і перевірка основних ідей проектованого продукту. У ряді випадків прототип дозволяє визначити здійсненність проекту. *Макетування, або прототипування*, – це процес створення моделі потрібного програмного продукту. Модель може приймати одну з трьох форм:

* макет на листку паперу, або на основі ПК (зображає людино-машинний діалог);
* працюючий макет (виконує деяку частину потрібних функцій);
* існуюча програма (характеристики якої потрібно покращити).

Макетування базується на багаторазовому повторенні ітерацій, в який беруть участь замовник і розробник. Воно починається зі збору й уточнення вимог до створюваного ПЗ. Розробник і замовник зустрічаються і визначають всі цілі ПЗ, встановлюють, які вимоги відомі, а які потрібно конкретизувати. Потім виконується швидке проектування, при якому зосереджуються на тих характеристиках ПЗ, які повинен бачити користувач. Швидке проектування приводить до побудови макету, який оцінюється замовником і використовується для уточнення вимог до ПЗ. Ітерації повторюються до тих пір, поки макет не виявить всі вимоги замовника і, тим самим, не дасть можливості розробнику зрозуміти, що потрібно робити.

Перевага макетування (прототипування): забезпечує визначення повних вимог до ПЗ. Недоліками макетування є те, що або замовник, або розробник може прийняти макет за готовий програмний продукт.

***Адаптивні моделі***

* Scrum;
* XP (Extreme Programming);
* Adaptive Software development (ASD);
* Dynamic System Development Model (DSDM);
* Feature Driven Development (FDD)

## Scrum

Scrum – один із основних підходів до гнучкої розробки ПЗ. Був розроблений Джефом Сазерлендом та Кеном Швабером в 1994 році.

Модель Scrum включає три стадії:

* підготовка;
* розробка запланованих функціональних можливостей;
* завершення.

Стадія *підготовки* ітерації включає в себе дві під стадії:

1. планування: розробка нового Product Log (список основних вимог до продукту, які мають бути реалізовані протягом виконання проекту), визначення дати кінцевого релізу та функціоналу для однієї або декількох ітерацій; вибір початкової ітерації; визначення команди розробників для ітерації; визначення можливих ризиків та засобів їх контролю; перегляд та можливе коректування Product Log; вибір та валідація інструментів та технології для розробки; визначення бюджету релізу, включаючи розробку, маркетинг, тренінги; узгодження з керівництвом бюджету та функціонального вмісту релізу.
2. високорівневий . дизайн: перегляд зазначених в Product Log специфікацій; визначення необхідних змін в специфікаціях для імплементації нових функціональних можливостей програмного продукту; проведення аналізу системи для виділення необхідних дій для створення, розширення чи оновлення існуючого ПЗ; уточнення архітектури системи для підтримки запланованих в Product Log специфікацій; виявлення проблем та питань, що можуть виникнути в процесі розробки; створення плану імплементації для кожної специфікації Product Log, обговорення та затвердження цих планів командою розробників.

*Розробка* – ітеративний процес розробки запланованих функціональних можливостей. Керівництво проекту визначає час, функціональне наповнення, якість кожної ітерації. По завершенню кожної ітерації відбувається аналіз отриманих результатів. Розробка включає наступні процеси:

* періодичне обговорення отриманих результатів та планів наступного релізу;
* перегляд та коригування стандартів, яким буде відповідати розроблений продукт;
* ітеративні спринти, поки програмний продукт буде готовий до випуску.

Спринт – сукупність усіх завдань, що будуть виконувати розробники протягом визначеного періоду (1-4 тижні). Довжина спринту залежить від складності продукту, оцінених ризиків. Кожен спринт включає всі традиційні фази розробки ПЗ: створення специфікації, аналіз, дизайн, розробка, тестування та демонстрація розроблених функціональних можливостей. Довжина спринту має бути фіксованою (30 днів).

*Стадія завершення* в Scrum містить усі необхідні завдання для остаточного випуску системи, після того, як закінчилася розробка усіх запланованих функціональних можливостей. Такі процеси як: системне тестування, написання документації, підготовка тренінгів.

Основні переваги моделі Scrum:

1. адаптивність до потреб ринку (додавання нових вимог/ зміна існуючих не призводить до серйозних змін в проекті);
2. короткі ітерації. В кінці кожної ітерації на виході готовий продукт, який може бути доставлений замовникам на перевірку/ нової версії продукту на ринок;
3. висока ефективність команд. Підвищується відповідальність кожного програміста за результат його роботи та проекту в цілому;
4. легкість в управління. Модель є прозорою, зрозумілою та легкою. Не потрібно використовувати важких інструментальних засобів для управління проектом.

Недоліки:

1. обмеження на розмір команди;
2. обмеження на місцезнаходження команди

## XP-програмування

Екстремальне програмування (eXtreme Programming, XP) – це полегшений (рухомий) процес проектування ПЗ (Кент Бек, 1999р.). XP-процес орієнтований на групи малого і середнього розміру, які проектують ПЗ в умовах невизначеності, або швидкої зміни вимог. XP-групу утворюють до 10 працівників, які розміщуються в одному приміщенні.

Основна ідея XP – зняти високу вартість змін, яка характерна для програм з використанням об’єктів, типових рішень і реляційних БД. XP-група має справу зі змінами вимог на протязі всього ітераційного циклу розробки, причому цикл складається з дуже коротких ітерацій. Чотирма базовими діями в XP-циклі є: кодування, тестування, вислуховування замовника і проектування. Динамізм забезпечується за допомогою чотирьох характеристик: безперервного зв’язку з замовником (і в межах групи), простоти (завжди вибирається мінімальне рішення), швидкого зворотного зв’язку (за допомогою модульного і функціонального тестування), сміливості в проведені профілактики можливих проблем.

Більшість принципів, які підтримуються XP (мінімальність, простота, еволюційний цикл розробки, мала тривалість ітерації, участь замовника, оптимальні стандарти кодування і т.д.) застосовуються в будь-якому впорядкованому процесі. Проте в XP ці принципи досягають екстремальних значень.

Базис ХР утворюють дванадцять методів.

1. Гра планування – швидке визначення області дії наступної реалізації шляхом об’єднання ділових пріоритетів і технічних оцінок. Замовник формує область дії, пріоритетність і терміни з точки зору бізнесу, а розробник оцінює і слідкує за прогресом проектування. Гра планування і часта зміна версій залежать від замовника, який забезпечує набір „історій” (коротких описів), що характеризують роботи, які будуть виконуватися для кожної версії системи. Версії генеруються кожні два тижні, тому розробники і замовник повинні досягнути погодження про те, які історії будуть реалізовані в межах двох тижнів. Повну функціональність характеризує пул історій; але для наступної двохтижневої ітерації з пула вибирається підмножина історій, що найбільш важлива для замовника. В будь-який час в пул можуть бути добавлені нові історії, таким чином, вимоги можуть швидко змінюватися. Проте процеси двохтижневої генерації базуються на найбільш важливих функціях, які входять в поточний пул, значить, зміни стають керованими. Локальний замовник забезпечує підтримку цього стилю ітераційної розробки.

2. Часта зміна версій – швидкий запуск в виробництво простої системи. Нові версії реалізуються в дуже короткому (двохтижневому) циклі.

3. Метафора – вся розробка проводиться на основі простої, загальнодоступної історії про те, як працює вся система. Метафора забезпечує глобальне „бачення” проекту. Вона могла б розглядатися як високорівнева архітектура, але ХР передбачає проектування при мінімумі проектної документації. Точніше кажучи, ХР пропонує неперервне перепроектування (за допомогою реорганізації), при якому немає потреби в деталізованій проектній документації, а для інженерів супроводу єдиним надійним джерелом інформації є програмний код. Часто після написання коду проектна документація викидається. Вона зберігається тільки в тому випадку, коли замовник тимчасово втрачає здатність придумувати нові історії. Тоді систему поміщають в „нафталін” і пишуть інструкцію сторінок на п’ять-десять за „нафталіновим” варіантом системи.

4. Просте проектування – проектування виконується настільки просто, наскільки це можливо в даний момент.

5. Тестування – неперервне написання тестів для модулів, які повинні виконуватися бездоганно; замовники пишуть тести для демонстрації закінченості функцій. „Тестуй, а потім кодуй” означає, що вхідним критерієм для написання коду є тестовий варіант, який дав відмову. Ця фраза виражає акцентування ХР- процесу на тестуванні. Вона відображає принцип, за яким спочатку планується тестування, а тестові варіанти розробляються паралельно аналізу вимог, хоча традиційний підхід полягає в тестуванні „чорного ящика”.

6. Реорганізація – система реструктурується, але її поведінка не змінюється; мета – усунути дублювання, покращити взаємодію, спростити систему або добавити в неї гнучкість. Використання реорганізації приводить до реалізації найпростішого рішення, яке задовольняє поточні потреби. Зміни в вимогах заставляють відмовитися від всіх „загальних рішень”.

7. Парне програмування – весь код пишеться двома програмістами, які працюють на одному комп’ютері. Це один із найбільш суперечливих методів в ХР, він впливає на ресурси, що важливо для менеджерів, які вирішують, чи буде проект використовувати ХР. Може здатися, що парне програмування подвоює ресурси, але дослідження показали: парне програмування приводить до підвищенню якості і зменшення часу циклу. Для узгодженої групи затрати збільшуються на 15%, а час циклу скорочується на 40-50%. Співробітництво покращує процес вирішення проблем, покращення якості суттєво знижує затрати на супровід, які перевищують вартість додаткових ресурсів на всьому циклі розробки.

8. Колективне володіння кодом означає, що будь-який розробник може змінити будь-який фрагмент коду системи в будь-який час.

9. Неперервна інтеграція – система інтегрується і будується багато разів на день, в міру завершення кожної задачі. Неперервне регресивне тестування, тобто повторення попередніх тестів, гарантує, що зміни вимог не приведуть до регресу функціональності. Неперервна інтеграція, неперервне регресивне тестування і парне програмування ХР забезпечують захист від проблем, які виникають при цьому.

10. 40-годинний тиждень – як правило, працюють не більше 40 годин на тиждень. Не можна збільшувати робочий тиждень за рахунок позаурочних робіт.

11. Локальний замовник – в групі весь час повинен знаходитися представник замовника, який дійсно готовий відповідати на питання розробників.

12. Стандарти кодування – повинні витримуватися правила, які забезпечують однакове представлення програмного коду в усіх частинах програмної системи.

## Adaptive Software development (ASD)

**Адаптивна розробка ПЗ**  (АРПЗ) — це процес розробки ПЗ, запропонований Джимом Хайсмітом та Семом Байером під час швидкої розробки програмних продуктів. Дана модель втілює в собі принцип безперервної адаптації процесу розробки, близького до нормального плину справ.

Адаптивна розробка замінює водоспадну модель повторюваними серіями обдумування, співробітництва та навчання. Цей динамічний цикл передбачає постійне навчання та адаптацію до виникаючих станів проекту. Він пов'язаний із постійними змінами, повторними оцінками, намаганнями передбачити невідоме на поточний момент майбутнє проекту і потребує тісного зв’язку між розробниками, тестувальниками і замовниками. Характеристиками ЖЦ АРПЗ є зосередженість уваги, ітеративність, обмеженість за часом, управління ризиком, терпимість до змін.

АРПЗ побудована на концептуальній базі теорії складних адаптивних систем. Вона розрахована на використання в екстремальних проектах, в яких превалюють швидкий темп розробок, непередбачуваність і часті зміни.

При АРПЗ звичайний статичний ЖЦ «**Планування-Проектування-Конструювання**» змінюється на динамічний «**Обдумування-Взаємодія-навчання**).

***Dynamic System Development Model (DSDM)***

Методологія DSDM з'явилася в результаті роботи консорціуму з 17 англійських компаній. Ціла організація займається розробкою посібників з цієї методології, організацією навчальних курсів, програм акредитації і т. інш.

За цією методологією все починається з вивчення здійсненності програми і області її застосування. На першому етапі потрібно зрозуміти, чи підходить DSDM для даного проекту. На наступному - вивчення області застосування програми; передбачається на короткій серії семінарів, де програмісти дізнаються про ту сферу бізнесу, для якої їм потрібно працювати. Тут же обговорюються основні положення, що стосуються архітектури майбутньої системи і план проекту.

Далі процес ділиться на три взаємопов'язаних цикли:

1. цикл функціональної моделі (відповідає за створення аналітичної документації та прототипів);
2. цикл проектування і конструювання (приведення системи в робочий стан);
3. цикл реалізації (забезпечує розгортання програмної системи).

***Feature Driven Development (FDD)***

Ця методологія була розроблена Джеффом Де Люка і Пітером Коад. Вона робить основний акцент на короткі ітерації, кожна з яких служить для опрацювання певної частини функціональності системи. Згідно FDD, одна ітерація триває два тижні.

FDD налічує п'ять процесів. Перші три з них відносяться до початку проекту.

1. Розробка загальної моделі
2. Складання списку необхідних властивостей системи
3. Планування роботи над кожним властивістю
4. Проектування кожної властивості
5. Конструювання кожної властивості

Останні два кроки необхідно робити під час кожної ітерації. При цьому кожен процес розбивається на завдання і має критерії верифікації.

Всі розробники діляться на два види: "Власники класу" (власники класів) і «головні програмісти" (старші програмісти). Старші програмісти - це найбільш досвідчені розробники. Саме їм доручається розробка конкретних властивостей системи. Однак вони не займаються цим самостійно: старший програміст визначає, які класи зайняті в реалізації даної конкретного властивості, після чого збирає команду з власників необхідних класів, яка і займатиметься розробкою. Сам він діє як координатор, головний проектувальник і керівник, а на частку власників класів залишається, здебільшого, безпосереднє кодування.