МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ  
ФАКУЛЬТЕТ КОМП’ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра економічної кібернетики

Комплексне практичне індивідуальне завдання

з дисципліни «Конструювання ПЗ»

на тему:

«Життєвий цикл програмного забезпечення»

Виконав :

студент групи ПЗС-32

Зварич Р. О.

Перевірив:

Сирник О. Й.

Тернопіль

2015

Зміст

[Вступ 2](#_Toc437271035)

[Загальні поняття 3](#_Toc437271036)

[Моделі життєвого циклу програмного забезпечення 4](#_Toc437271037)

[Висновки 7](#_Toc437271038)

[Список використаних джерел 8](#_Toc437271039)

# Вступ

Життє́вий цикл програ́много забезпе́чення — сукупність окремих етапів робіт, що проводяться у заданому порядку протягом періоду часу, який починається з вирішення питання про розроблення програмного забезпечення і закінчується припиненням використання програмного забезпечення.

Поняття життєвого циклу програмного забезпечення відносять до дисципліни "Програмна інженерія".

# Загальні поняття

В загальному випадку, життєвий цикл визначається моделлю й описується у формі методології (методу). Модель або парадигма життєвого циклу визначає загальну організацію і, як правило, основні його фази та принципи переходу між ними. Методологія (метод) визначає комплекс робіт, їх детальний зміст і рольову відповідальність спеціалістів на всіх етапах вибраної моделі.

Життєвий цикл програмного забезпечення супроводжується розробленням, обігом та використанням програмної документації.

Програмна документація — сукупність документів, що містять відомості, необхідні для розробки, виготовлення, супроводу та експлуатації програм. Програмна документація є одним з видів технічної документації.

Комплекс державних стандартів, що встановлюють взаємопов'язані правила розробки, оформлення та обігу програм і програмної документації називається «Єдина система програмної документації» (ЄСПД).

# Моделі життєвого циклу програмного забезпечення

Модель життєвого циклу - це структура, що складається із процесів, робіт та задач, які включають в себе розробку, експлуатацію і супровід програмного продукту; охоплює життя системи від визначення вимог до неї до припинення її використання. На сьогодні найбільшого розповсюдження набули дві моделі:

каскадна модель;

спіральна модель.

Каскадна модель

Однією з перших з’явилась каскадна модель, в якій кожен етап роботи виконується лише раз. На кожному етапі робота виконується настільки ретельно, щоб потреби повертатись до попереднього не виникало. Результат виконання кожного етапу, перед передачею в наступний, піддається верифікації.

Спіральна модель

Розробка ітераціями відображає об'єктивно існуючий спіральний цикл створення системи. Неповне завершення робіт на кожному етапі дозволяє переходити на наступний етап, не чекаючи повного завершення роботи на поточному. При ітеративному способі розробки відсутню роботу можна буде виконати на наступній ітерації. Головне ж завдання - щонайшвидше показати користувачам системи працездатний продукт, тим самим активізуючи процес уточнення і доповнення вимог.

Виходячи з можливості внесення змін, як в процес, так і в проміжний продукт було створено спіральну модель ЖЦ (рис.3).

Внесення змін орієнтоване на задоволення потреби користувачів одразу, як тільки буде встановлено, що створені артефакти або елементи документації не відповідають дійсному стану розробки.

Дана модель ЖЦ допускає аналіз продукту на витку розробки, його перевірку, оцінку правильності та прийняття рішення про перехід на наступний виток або повернення на попередній виток для доопрацювання на ньому проміжного продукту.

Відмінність цієї моделі від каскадної полягає в можливості багато разів повертатися до процесу формулювання вимог і до повторної розробки версії системи з будь-якого процесу моделі.

Для програмного продукту така модель не дуже підходить з декількох причин. По-перше, висловлення вимог замовником носить суб'єктивний характер, вимоги можуть багаторазово уточнюватися протягом розробки ПС і навіть після завершення та випробовування, і часом може з'ясуватися, що замовник «хотів зовсім інше». По-друге, змінюються обставини та умови використання системи, тому загальновизнаним законом програмної інженерії є закон еволюції, який сформулюємо так: кожна діюча ПС з часом потребує внесення змін або виводиться з експлуатації.

При необхідності внесення змін до системи на кожному витку з метою отримання нової версії системи обов'язково вносяться зміни в заздалегідь зафіксовані вимоги, після чого повертаються на попередній виток спіралі для продовження реалізації нової версії системи з урахуванням усіх змін.

Еволюційна модель

Використання еволюційної моделі припускає проведення дослідження предметної області для вивчення потреб її замовника і аналізу можливості застосування цієї моделі для реалізації. Модель використовується для розробки нескладних і некритичних систем, де головною вимогою є реалізація функцій системи. При цьому вимоги не можуть бути визначені відразу і повністю. Тому розробка системи здійснюється ітераційним шляхом її еволюційного розвитку з отриманням деякого варіанта системи–прототипу, на якому перевіряється реалізація вимог. Іншими словами, такий процес за своєю суттю є ітераційним, з етапами розробки, що повторюються, починаючи від змінених вимог і закінчуючи отриманням готового продукту. В деякому розумінні до цього типу моделі можна віднести спіральну модель.

У літературі вона часто називається моделлю швидкої розробки програм RAD (Rapid Application Development).

У даній моделі наведені дії, які пов'язані з аналізом її застосовності для конкретного виду системи, а також обстеженням замовника для визначення потреб користувача при розробці плану створення прототипу.

У моделі є дві головні ітерації розробки функціонального прототипу, проектування і реалізації системи з метою перевірки, чи задовольняє вона всі функціональні і нефункціональні вимоги. Основною ідеєю цієї моделі є моделювання окремих функцій системи в прототипі і поступове еволюційне його доопрацювання до виконання всіх заданих функціональних вимог.

Ітерацій з отримання проміжних варіантів прототипу може бути декілька, в кожній з яких додається функція і повторно моделюється робота прототипу. І так до тих пір, поки не будуть промодельовані всі функції, задані у вимогах до системи. Після цього виконується ще одна ітерація – остаточне програмування для отримання готової системи.

Ця модель застосовується для систем, в яких найбільш важливими є функціональні можливості, і які необхідно швидко продемонструвати на CASE-засобах.

Оскільки проміжні прототипи системи відповідають реалізації деяких функціональних вимог, їх можна перевіряти і під час супроводу і експлуатації, тобто разом з процесом розробки чергових прототипів системи. При цьому допоміжні і організаційні процеси можуть виконуватися разом з процесом розроблення і накопичувати відомості за даними кількісних і якісних оцінок на процесах розроблення.

При цьому враховуються такі чинники ризику:

– реалізація всіх функцій системи одночасно може призвести до громіздкості;

– обмежені людські ресурси зайняті розробкою протягом тривалого часу.

Переваги застосування даної моделі ЖЦ такі:

– швидка реалізація деяких функціональних можливостей системи і їх апробація;

– використання проміжного продукту в наступному прототипі;

– виділення окремих функціональних частин для реалізації їх у вигляді прототипу;

– можливість збільшення фінансування системи;

– зворотний зв'язок із замовником для уточнення функціональних вимог;

– спрощення внесення змін у зв'язку із заміною окремих функції.

Модель розвивається у напряму додавання нефункціональних вимог до системи, пов'язаних із захистом і безпекою даних, несанкціонованим доступом до них і ін.

# Висновки

Ознайомившись з інформацією про систему контрою версій можна сказати – що це потужний інструмент в руках розробника програмного забезпечення. За допомогою контролю версій зміни в програмі відслідковувати легше, це допомагає також у командній роботі.

Можна припустити, що більш за все саме командам розробників доцільно використовувати системи контролю версій. А як відомо – комп’ютерні ігри створюються великою кількістю людей, часом це понад 200 розробників, які працюють над проектом одночасно. Коли один член команди працює над своєю ділянкою роботи, а інший над своєю, то в кінці-кінців можуть виникнути різні складні ситуації. Для того, щоб у процесі розробки незначні помилки не призводили до збоїв у всій системі і використовуються системи контролю версіями.

# Список використаних джерел

1. <http://pidruchniki.com/10050711/informatika/modeli_zhittyevogo_tsiklu_programnogo_zabezpechennya>
2. <http://www.programsfactory.univ.kiev.ua/content/books/2/53>
3. <http://www.programsfactory.univ.kiev.ua/content/books/2/50>
4. <http://web.archive.org/web/20130903043848/http://ami.lnu.edu.ua/books/ami/ООМ.pdf>