**Лабораторна робота №13. Застосування технологій конструювання програмного забезпечення.**

**Мета роботи**. Навчитися застосовувати на практиці знання щодо застосування технологій конструювання програмного забезпечення.

**Завдання на лабораторну роботу**

1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

2. Для власного курсового проекту потрібно визначити методи, засоби і продукти, які мають використовуватися для проекту, а саме:

* визначте факти, що визначають специфіку ПЗ Вашого проекту як продукту;
* визначте перелік національних стандартів, відповідно до яких виконується розроблення Вашої курсової, та вкажіть призначення використання стандарту для Вашої курсової роботи;
* визначте та обґрунтуйте використання моделі життєвого циклу та стратегія конструювання ПЗ щодо Вашої курсової роботи;
* визначте та обґрунтуйте використання обраної методології розробки ПЗ;
* визначте та обґрунтуйте причини обрання інструментарію розроблення ПЗ;
* визначте перелік процедур, які застосовуються на технологічних етапах розробки ПЗ (наприклад, на етапі проектування застосовуються діаграми класів та компонентів для опису структури системи ).

3. Визначити технологічний процес розроблення програмного продукту Вашої курсової роботи у вигляді відповідних UML – діаграм.

4. Оформити звіт, де надати у вигляді таблиць результати за пп.2-3, додавши відповідні пояснення щодо обраного рішення.

**Контрольні запитання**

1. Що таке технології програмування?
2. Які моделі життєвого циклу дозволяють оперативно реагувати на зміну вимог до ПЗ?
3. Які ключові точки проекту розроблення ПЗ вам відомі?
4. Як пов’язані між собою строки проектних робіт, їх обсяг та вартість? Яким чином, керуючи цими показниками, можна забезпечити якість проекту?
5. Які складові методології розроблення потрібно враховувати при її виборі?

Результати надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді текстових файлів з іменем у форматі **KPZ<Номер групи><Номер лекції / лабораторної>[-<Номер завдання>][літера позначення типу роботи L – лекція, R - лабораторна]<Прізвище англійською>**. Наприклад, **KPZ4112R**buts.doc. Відповіді повинні бути не довгими і змістовними. Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності відповідей-"близнюків" відповідь буде зараховуватися першому за часом надсилання.

**Строк відсилки ЛР ІПЗ-41**

**ІПЗ-42 .**

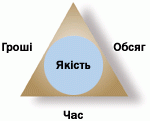
**ТЕОРЕТИЧНІ ВІДОМОСТІ**

Технологія конструювання програмного забезпечення (ТКПЗ) - система інженерних принципів для створення ПЗ, відповідно, діяльність під час розроблення ПЗ, як і в іншій інженерній галузі, складається з виконання операцій і проектів, на виконання яких виділяються обмежені ресурси.

Головна відмінність операцій від проектів полягає в тому, що операції виконуються постійно і повторюються, тоді як проект тимчасовий і унікальний. Виходячи з цього, ***проект*** визначається як тимчасове зусилля, розпочате для створення унікального продукту чи послуги. «Тимчасове» означає, що кожен проект має точно визначені дати початку та закінчення. Говорячи про унікальність продукту, ми маємо на увазі, що вони мають помітні відмінності від усіх аналогічних продуктів або послуг. Таким чином, розроблення ПЗ відповідає визначенню проекту і для організації цього процесу можна застосовувати методи та інструментарій управління проектами. Наприклад, розроблення веб-сайту є проектом, тоді як підтримка його впродовж тривалого часу – це операційна діяльність.

У кожного проекту є чітко визначені початок і кінець. Кінець проекту настає разом із досягненням усіх його цілей або коли стає зрозумілим, що ці цілі не будуть або не можуть бути досягнуті. Тимчасовість не означає короткостроковість проекту – розроблення складної програмної системи може тривати кілька років, хоча, як правило проекти мають обмежені часові рамки для створення ПЗ, оскільки сприятлива для них ситуація на ринку складається на обмежений час. Крім того, проектна команда після його закінчення розпадається, а її члени переходять в інші проекти.

Проект дуже часто плутають із програмою, тобто координованим управлінням групою проектів всередині однієї організації. Управління відразу декількома проектами скоординоване для того, щоб отримати вигоду, яку неможливо одержати від окремого управління кожним із них.

Проект виконується для досягнення певного результату в певні терміни і за певні гроші. Це тріо *часу (time), бюджету (cost) і обсягу робіт (scope)* часто називають ***проектним,*** або залізним, ***трикутником*** (рис. 1) [8], оскільки при внесенні змін в один із цих елементів змінюються інші. Хоча для проекту однаковою мірою важливі всі три елементи, як правило, тільки один із них залежно від пріоритетів має найбільший вплив на інші.

Те, як зміни у плані впливають на інші сторони трикутника, залежить від обставин і специфіки проекту. У деяких випадках зменшення терміну збільшує вартість, а в інших - зменшує.

Рисунок 1 Проектний трикутник *.*

*Якість (quality)* – це четвертий елемент проектного трикутника. Вона знаходиться у центрі, і будь-що зміна сторін впливає на неї. Наприклад, якщо керівникові вдалося знайти додатковий час у розкладі, то можна збільшити обсяг робіт, додавши завдання і збільшивши тривалість проекту. Ці додаткові завдання і час дозволяють домогтися більш високого рівня якості проекту і виробленого продукту.

Якщо ж потрібно знизити витрати, щоб вкластися в бюджет, можливо, знадобиться зменшити обсяг робіт, прибравши деякі із завдань або зменшивши їх тривалість. Зі зменшеним обсягом робіт у проекті буде менше шансів вийти на необхідний рівень якості, тому зниження витрат може призвести до погіршення якості проекту.

План проекту складається для того, щоб визначити, за допомогою яких робіт буде досягатися результат проекту, які люди й устаткування потрібні для виконання цих робіт і в що час ці люди й устаткування будуть зайняті роботою в проекті. Тому проектний план містить три основні елементи: завдання, ресурси і призначення.

**Завдання (Tasks)**

Завданням є робота, здійснювана у рамках проекту для досягнення певного результату. Наприклад, у проекті створення програми для автоматизації формування документації підприємства завданням буде Створення шаблонів. Оскільки, як правило, проект містить багато завдань, то для зручності відстеження плану їх об'єднують у групи, або фази. Сукупність фаз проекту називається його життєвим циклом.

**Фази (Summary tasks)**

Фаза проекту складається з одного або кількох завдань, у результаті виконання яких досягається один або кілька основних результатів проекту. Таким чином, результати, досягнуті завдяки виконанню кожної із задач, що входять у фазу, формують її результат. Фази можуть складатися як із завдань, так і з інших фаз.

Якщо для досягнення результатів завдання потрібно виконати тільки її, то для досягнення результату фази потрібно виконати групу інших завдань.

При плануванні робіт потрібно пам'ятати, що чим детальніше буде план проекту, тим точніше він буде відповідати реальній ситуації. У тих випадках, де це можливо, варто розбивати великі завдання на підзадачі (перетворювати завдання в фази). Формальними критеріями, які показують, що завдання можна розбити на підзадачі, є тривалість (завдання рідко тривають довше 2-3 днів) і велике число задіяних виконавців (як правило, якщо над вирішенням завдання працюють більше 2-3 осіб, то кожен вирішує свою власну задачу, яку можна окремо врахувати у плані проекту).

**Завершальні завдання**

Завдання, у результаті виконання яких досягаються проміжні цілі, називаються завершальними завданнями.

**Тривалість (Duration) і трудовитрати (Work)**

Тривалість завдання - це період робочого часу, що необхідний для того, щоб виконати її.

Тривалість може не відповідати трудовитратам співробітника, що займається завданням. Тривалість відповідає часу, через що буде отриманий результат задачі, а трудовитрати – часу, витраченому співробітниками на отримання результату.

**Залежності (Dependencies) та зв'язки (Links)**

Завдання у плані проекту взаємозв'язані. Наприклад, часто одна задача не може початися, поки не закінчена інша (тестування модуля не може початися раніше написання його коду).

На плані проекту залежності позначаються за допомогою зв'язків. Обидва ці терміни – залежність і зв'язок – використовуються з одним і тим самим змістом і позначають логіку, певну послідовність робіт у плані проекту.

**Ролі (Roles) і ресурси (Resources)**

Під ресурсами розуміють співробітників та обладнання, які необхідні для виконання проектних завдань.

Кожен співробітник, що бере участь у проекті, отримує певну роль відповідно до своєї кваліфікації, вимог проекту та регламентів, що діють в організації. Наприклад, в одному проекті співробітник може виступати в ролі архітектора додатків, а в іншому той самий працівник може бути задіяний у ролі програміста. При складанні списку ресурсів часто використовується рольове планування. Наприклад, спочатку визначається, що для виконання робіт потрібні три програмісти і один менеджер, а потім, коли план проекту затверджений, вибираються конкретні співробітники для участі в цих ролях.

**Призначення (Assignments)**

Призначення - це зв'язок певної задачі і ресурсів, необхідних для її виконання. При цьому на одну задачу можуть бути призначені декілька ресурсів, як матеріальних, так і нематеріальних.

Призначення об'єднують у плані ресурси і завдання, роблячи план цілісним. Завдяки призначенням вирішується цілий ряд завдань планування. По-перше, визначаються відповідальні за виконання завдань. По-друге, коли визначені завдання, за які відповідає ресурс, можна розрахувати загальний обсяг часу, що витрачається ним на проект, а отже, його вартість для проекту. По-третє, визначивши вартість участі всіх ресурсів у проекті, можна підрахувати його загальну вартість. Нарешті, призначаючи ресурси на завдання, можна скорочувати термін виконання робіт, виділяючи на них більше ресурсів і тим самим скорочуючи загальну тривалість проекту.

***Основні форми планів робіт***

Результатом фази оцінки здійсненності є детальна специфікація (технічне завдання), план роботи та оцінка вартості. Найбільш традиційними формами плану можна вважати *мережевий графік та діаграму Ганта (Gantt diagram)*. Діаграма Ганта дозволяє визначити, що частина робіт повинна бути виконана у кожен момент часу, тому її частіше використовують керівники проектів, тоді як технічні фахівці віддають перевагу мережевим графікам, оскільки вони містять інформацію про тривалість кожного виду роботи.

***Мережевий графік*** подається у вигляді орієнтованого графа з двома виділеними вершинами - початок і кінець роботи (рис. 2). Вершинами графа є події, що відповідають пунктам плану, а ребрами - роботи. Події повинні виражатися дієсловами доконаного виду: "тексти програм передані в базу вхідних даних", "усі тести пропущені", "група оцінки якості дала позитивний висновок" тощо. Ребра навантажуються оцінками тривалості робіт, наприклад, у днях або тижнях. *Переваги* використання мережевого графіка при плануванні робіт полягають у такому:

* на ньому чітко видно залежність робіт одна від одної;
* на основі мережевого графіка можна обчислити тривалість усієї роботи;
* користуючись результатами цих розрахунків, можна оптимізувати тривалість і витрати на роботу.

Тривалість робіт обчислюється так: підсумовуємо тривалості робіт за всіма можливими шляхами в графі. Той шлях від початку до кінця, що є найдовшим, оголошується ***критичним***, оскільки затримка будь-якої роботи, що лежить на цьому шляху, приводить до затримки всієї роботи в цілому. Зрозуміло, що критичних шляхів може бути декілька.

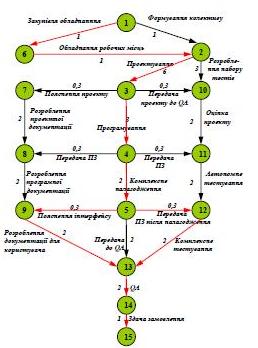


Рисунок 2 – Мережевий графік робіт при розробленні ПЗ

Ще однією популярною формою графічного представлення плану робіт є діаграма Ганта. ***Діаграма Ганта*** являє собою прямокутник: зліва направо рівномірно відлічуються періоди часу (тижні, місяці), зверху вниз перераховуються роботи, причому кожна робота представляється у вигляді відрізка, початок і кінець якого розміщуються у відповідному періоді (рис. 3). *Переваги* використання діаграм Ганта при плануванні робіт полягають у тому, що чітко видно, що виконується кожного тижня, а також зручно позначати кількість виконавців роботи.

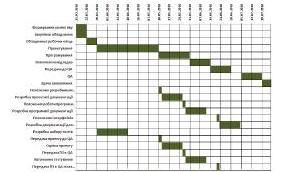


Рисунок 3 – Діаграма Ганта

***Керування та організація робіт***

Особливості процесу керування визначаються в першу чергу набором методів та технологій, які будуть використовуватися під час розроблення ПЗ. Цей вибір виконує керівник проекту, і обрана ним методологія в першу чергу визначається його особистим досвідом, а потім вимогами замовника. Алістер Коберн визначає такі **принципи вибору методології розроблення**. *Перший принцип* визначає, що велика за розміром методологія потрібна великим командам розробників. Розмір методології визначається кількістю елементів керування у проекті, до яких належать види діяльності, застосовані стандарти, створювані артефакти, показники якості й інші характеристики. На рис.4 наведені складові методології.



Рисунок 4 – Складові методології

1. ***Групи*** *(Teams)* – розподіл розробників за групами із визначенням їх ролей (наприклад, дизайнери, розробники документації, тестери та ін.).
2. ***Ролі*** *(Roles)* – визначення посади, що наводиться в оголошенні про роботу (наприклад, менеджер проекту, тестувальник, розробник інтерфейсу та ін.).
3. ***Уміння*** *(Skills)* – навички, які повинні мати претенденти на посаду.
4. ***Види діяльності*** *(Activities)* – зустрічі, відгуки, ключові точки (milestones) та інша діяльність, яку виконує або ініціює розробник.
5. ***Техніки*** *(Techniques)* – техніки, які розробники використовують (наприклад, Java-програмування, моделювання варіантів використання та ін.).
6. ***Інструменти*** *(Tools)* – інструменти, які використовують розробники, відповідно до обраних технік та стандартів.
7. ***Якість*** *(Quality)* – правила, запитання або зауваження, які необхідно відстежувати для кожного результату
8. ***Продукти*** *(Work Products)* – результат, що передають розробники або групи іншим розробникам або групам (наприклад, варіанти використання, визначення класів, визначення тестів, документація, визначення інтерфейсів та ін.).
9. ***Стандарти*** *(Standards*) – що дозволяється або не дозволяється для продуктів. Розрізнюють стандарти позначень, у тому числі і мови програмування, стандарти керування та прийняття рішень та домовленості проекту. Деякі методології обирають не всі стандарти одразу, а відкладають визначення деяких до певного етапу проекту.

Додатковою характеристикою методології також є ***Цінності*** *(Values)* – те, до чого прагне команда, які види комунікації та організації роботи переважають. Цінності визначають ефективність методології: одна й та сама методологія для команд із різними цінностями буде мати відмінні показники ефективності.

Потрібно зауважити, що проекти з використанням полегшених методологій (тих, у яких мало уваги приділяється формалізації робіт) частіше приходять до успіху, ніж проекти із дуже формалізованими процедурами.

*Другий принцип* полягає у тому, що рівень критичності системи визначає «щільність» методології. Щільність методології визначається рівнем деталізації та взаємодії при її використанні. Більш щільні методології є більш формалізованими. А.Коберн пропонує розділяти системи на чотири групи залежно від рівня можливих втрат:

1. втрата комфортності *– найнижча критичність* – наприклад, збій у роботі модуля автоматизованого проектування призведе до збільшення ручних розрахунків;
2. втрата незначної суми матеріальних ресурсів, наприклад, неправильне нарахування заробітної плати автоматизованою системою можна виправити у ручному режимі;
3. матеріальні втрати не можна відшкодувати, наприклад, збій системи національного банку країни;
4. втрата життя – *найвища критичність* – наприклад, збій системи керування на атомній електростанції. Також до систем найвищої критичності відносять аерокосмічні системи, системи керування польотами.

Тобто для підвищення надійності систем із високим рівнем критичності потрібно розробляти детальний опис, проекту та докладну програмну документацію, регулярно надавати розроблені артефакти як для внутрішної перевірки, так і для перевірки замовниками. Усі ці заходи підвищують вартість артефактів, але зменшують кількість помилок ще на ранніх стадіях проекту.

*Третій принцип* визначає те, що збільшення розміру методології та/або її щільності приводить до збільшення вартості проекту. Зрозуміло, що координація різних груп розробників для передачі інформації, оновлення документації після перевірок потребують часу і зусиль для якісного цілісного сприйняття. Пояснюючи цей принцип, А.Коберн зауважує, що розмір проекту та розмір методології знаходяться у відношенні зворотної пропорції – для невеликої команди розробників не потрібна дуже формалізована методологія, бо вони легко можуть контактувати між собою та передавати будь-які артефакти. Головна проблема полягає у тому, що на початку проекту неможливо точно уявити його обсяги.

*Четвертий принцип* визначає, що найкращий спосіб комунікації між розробниками – безпосереднє спілкування. Цей принцип не абсолютизує вимогу бути усім розробникам в одній кімнаті, а надає рекомендації щодо підвищення ефективності роботи невеликої команди.

**Фундаментальні основи конструювання програмного забезпечення та методи вирішення**

Фундаментальні основи конструювання програмного забезпечення включають:

* Мінімізація складності
* Очікування змін
* Конструювання з можливістю перевірки
* Стандарти у конструюванні

Зменшення складності у конструюванні програмного забезпечення досягається за допомогою звертання особливої уваги на створення простого коду, який легко читається, іноді на шкоду прагненню зробити його ідеальним (наприклад, з точки зору гнучкості або слідування тим чи іншим уявленням про красу, витонченість коду, вправність тих чи інших прийомів тощо). Це не означає, що повинно обмежуватися застосування тих чи інших розвинених мовних можливостей використовуваних засобів програмування. Мається на увазі "лише" надання більшої значимості читаності коду, простоті тестування, прийнятного рівня продуктивності та задоволенню заданих критеріїв, замість постійного вдосконалення коду, не оглядаючись на терміни, функціональність і інші характеристики та обмеження проекту.

Мінімізація складності досягається, зокрема, за рахунок слідування стандартам, використанням низки специфічних технік кодування і підтримкою практик, спрямованих на забезпечення якості в конструюванні.

Очікування змін є однією з рушійних сил конструювання програмного забезпечення, оскільки більшість програмних систем змінюються з плином часу. Програмне забезпечення не є ізольованим від зовнішнього оточення (як системного, так і з точки зору галузі діяльності, для автоматизації задач і проблем якого воно застосовується). Більш того, програмні системи є частиною середовища, що змінюється, і повинні змінюватися разом з нею, а, іноді, і бути джерелом змін самого середовища. Очікування змін підтримується рядом технік кодування.

Конструювання для перевірки припускає, що побудова програмних систем повинна вестися таким чином, щоб сама програмна система допомагала вести пошук причин збоїв, будучи прозорою для застосування різних методів перевірки (і, до речі, внесення необхідних змін), як на стадії незалежного тестування (наприклад, інженерами-тестувальниками), так і в процесі операційної діяльності - експлуатації, коли особливо важлива можливість швидкого виявлення та виправлення помилок що виникають.

Серед технік, спрямованих на досягнення такого результату конструювання:

* Огляд, оцінка коду (code review)
* Модульне тестування (unit-testing)
* Структурування коду для і спільно з застосуванням автоматизованих засобів тестування (automated testing)
* Обмежене застосування складних або важких для розуміння мовних структур.

Стандарти, які безпосередньо застосовуються при конструюванні, включають:

* Комунікаційні методи (наприклад, стандарти форматів документів і оформлення вмісту)
* Мови програмування і відповідні стилі кодування (наприклад, Java Language Specification, що є частиною стандартної документації JDK - Java Development Kit і Java Style Guide, що пропонує загальний стиль кодування для мови програмування Java)
* Платформи (наприклад, стандарти програмних інтерфейсів для викликів функцій операційного середовища, такі як прикладні програмні інтерфейси платформи Windows - Win32 API, Application Programming Interface або .NET Framework SDK, Software Development Kit)
* Інструменти (не в термінах середовищ розробки, але можливих засобів конструювання - наприклад, UML як один зі стандартів для визначення нотацій для діаграм, що представляють структура коду і його елементів або деяких аспектів поведінки коду).

Конструювання залежить від зовнішніх стандартів, пов'язаних з мовами програмування, використовуваним інструментальним забезпеченням, технічними інтерфейсами і взаємним впливом конструювання програмного забезпечення та інших галузей знань програмної інженерії (в тому числі, пов'язаних дисциплін, наприклад, управління проектами). Стандарти створюються різними органами, наприклад, консорціумом OMG - Object Management Group (зокрема. Стандарти CORBA, UML, MDA,), міжнародними організаціями з стандартизації такими, як ISO/IEC, IEEE, TMF, виробниками платформ, операційних середовищ і т.д. (Наприклад, Microsoft, Sun Microsystems, CISCO, NOKIA), виробниками інструментів, систем управління базами даних і т.п. (Borland, IBM, Microsoft, Sun, Oracle, Розуміння цього факту дозволяє визначити достатній і повний набір стандартів, які застосовуються у проектній команді або організації в цілому.

*Література*

1. Лавріщева К.М. Програмна інженерія. Електронний підручник. URL: <http://csc.knu.ua/uk/library/books/lavrishcheva-6.pdf>.
2. Соммервил И. Инженерия программного обеспечения / И.Соммервил.–М. : Издательский дом «Вильямс», 2002. –623 с.
3. ДСТУ ISO/IEC 90003:2006. Програмна інженерія. Настанови щодо застосування ІSO 9001:2000 до програмного забезпечення (ІSO/ІЕС 90003:2004, IDT) - К.: Держстандарт України, 2006.
4. ДСТУ ISO/IEC TR 12182:2004. Інформаційні технології. Класифікація програмних засобів (ISO/IEC TR 12182:1998, IDT) - К.: Держстандарт України, 2004.
5. ДСТУ ISO/IEC 14598-1:2004. Інформаційні технології. Оцінювання програмного продукту. Частина 1. Загальний огляд (ISO/IEC 14598-1:1999, IDT) - К.: Держстандарт України, 2004.
6. ДСТУ ISO/IEC 15288:2005. Інформаційні технології. Процеси життєвого циклу системи (ISO/IEC 15288:2002, IDT) - К.: Держстандарт України, 2005.
7. . ДСТУ ISO/IEC 15939:2008. Інженерія систем і програмних засобів. Процес вимірювання. - К.: Держстандарт України, 2008.
8. Microsoft. Проектний трикутник. URL: <https://support.microsoft.com/uk-ua/office/%d1%82%d1%80%d0%b8%d0%ba%d1%83%d1%82%d0%bd%d0%b8%d0%ba-%d0%bf%d1%80%d0%be%d0%b5%d0%ba%d1%82%d1%83-8c892e06-d761-4d40-8e1f-17b33fdcf810?ui=uk-ua&rs=uk-ua&ad=ua>