**Лабораторна робота №2 Планування конструювання програмного забезпечення**

**Мета роботи**: засвоєння навичок планування розробки проектів програмного забезпечення

**Завдання:**

Продовжуємо працювати з проектом, визначеним у попередній ЛР.

1. Визначити вимоги до проекту.
2. Зробити попередні визначення щодо обрання методології реалізації проекту з відповідним обґрунтуванням.
3. Розбити проект на кілька етапів і для цих етапів скласти загальний календарний графік для своєчасного завершення розробки ПЗ.
4. Розробити перелік робіт по кожному етапу, що потрібно виконати для здійснення розробленого плану і задоволення визначених вимог.
5. По кожному етапу визначити наявні ресурси.
6. Виконати деталізацію плану та здійснити розподіл ресурсів для виконання робіт у встановлені терміни.
7. Розроблені вимоги, пропозиції щодо методології реалізації проекту, перелік робіт та визначені наявні ресурси, деталізований план (діаграма Ганта) внести до звіту, створеного на попередній ЛР. Для графічного представлення деталізації плану робіт використати діаграму Ганта, яку створювали в 6 семестрі по ОПІ (ОПІ ЛР 10 Управління програмними проектами Діаграма Ганта.doc).
8. Оформити роботу відповідно до встановлених вимог та здати на перевірку викладачеві, надіславши електронною поштою на адресу [t.i.lumpova@gmail.com](mailto:t.i.lumpova@gmail.com). Якщо викладач знаходить помилки чи неточності, він може повернути роботу на доопрацювання.

Файл з роботою повинен мати назву в такому форматі:

**KPZ <Номер групи><Номер лекції / лабораторної> [літера позначення типу роботи L – лекція, R – лабораторна]<Прізвище англійською>**.. Наприклад, **KPZ4101R**buts.doc.

Тему в заголовку листа записати

**KPZ<Номер групи>-ЛР<Номер лабораторної>-<Прізвище англійською>**

**Строк виконання цієї роботи ІПЗ-41 – 18.09.2021**

**ІПЗ-42 – 24.09.2021**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, Тему в заголовку листа записати

**KPZ <Номер групи>-Запитання-<Прізвище >**.

**Теоретичні відомості:**

Планування – вид діяльності, пов’язаний з постановкою цілей (задач) і дій в майбутньому. Планування, в загальному вигляді, має на увазі виконання наступних етапів:

* Ідентифікація цілей і задач
* Складання програми дій
* Виявлення необхідних ресурсів і їх джерел
* Визначення безпосередніх виконавців і доведення плану до них.

Планування – оптимальне розподілення ресурсів для досягнення поставленої мети. При плануванні використовуються моделі очікувань, які засновані на зусиллях пов'язаних з обсягом завдання. Ці моделі часто визначається як людино-місяці до функціональних точках.

З точки зору розподілу ресурсів розглядаються завдання, для яких призначені терміни і пов'язані з обладнанням і людьми. На цьому етапі планування зручно використовувати діаграми Ганта.

На цьому ж етапі визначаються правила і політика припинення проекту. При складанні плану проекту, необхідно визначити як проект буде управлятися і його план (звітність, моніторинг і контроль).

При виконанні планування КПЗ необхідно:

1. Виправити і затвердити вимоги.
2. Створити список робіт з описом модулів ПЗ.
3. Визначити необхідні ресурси людські та технічні
4. Зробити деталізацію плану виконання робіт.

Планування конструювання також визначає порядок, у якому компоненти створюються та інтегруються, стратегію інтегрування (наприклад, поетапну чи інкрементну інтеграцію), управління якістю програмного забезпечення, розподіл завдань між спеціалістами у програмній інженерії та інші завдання згідно обраного методу.

Різні типи проектів вимагають різних підготовки та конструювання. У таблиці 1 наведено три популярних типи проектів, а також оптимальні в більшості випадків методи роботи над ними.

*Таблиця 1*

| Тип ПЗ | Бізнес-системи | Системи цільового призначення | Вбудовані системи, від яких залежить людське життя |
| --- | --- | --- | --- |
| Типові застосунки | Інтернет-сайти. Сайти в інтрамережах. Системи управління матеріально-технічним постачанням. Ігри. Системи керування інформацією. Системи виплати заробітної плати. | Вбудоване ПЗ. Інтернет-сайти. Ігри. Пакетне ПЗ. Програмні інструменти. Web-сервіси. | Авіаційне ПЗ. Вбудоване ПЗ. ПЗ для медичних пристроїв. Операційні системи. Пакетне ПЗ. |
| Моделі життєвого циклу | Еволюційне прототипування. Гнучка розробка (екстремальне програмування, методологія Scrum та ін.). | Поетапна поставка. Еволюційна поставка. Спіральна розробка. | Поетапна поставка. Еволюційна поставка. Спіральна розробка. |
| Планування та управління | Інкрементне планування об’єкту. Планування тестування та гарантії якості при необхідності. Неформальний контроль над змінами. | Базове завчасне планування. Базове планування тестування. Планування гарантії якості при необхідності. Формальний контроль над змінами. | Вичерпне завчасне планування. Вичерпне планування тестування. Вичерпне планування гарантії якості. Ретельний контроль над змінами. |
| Вироблення вимог | Неформальна специфікація вимог | Напівформальна специфікація вимог. Огляди вимог при необхідності. | Формальна специфікація вимог. Формальні інспекції вимог. |
| Проектуван-ня | Комбінація проектування та кодування | Проектування архітектури. Неформальне детальне проектування. Огляди проекту при необхідності. | Проектування архітектури. Формальні інспекції архітектури. Формальне детальне проектування. Формальні інспекції детального проекту. |
| Конструюван-ня | Парне або індивідуальне програмування. Неформальна процедура реєстрації коду або її відсутність. | Парне або індивідуальне програмування. Неформальна процедура реєстрації коду. Огляди коду при необхідності. | Парне або індивідуальне програмування. Формальна процедура реєстрації коду. Формальні інспекції коду. |
| Тестування та гарантія якос-ті | Розробники тестують власний код. Попередня розробка тестів. Тестування окремої групи проводиться в малому об’ємі або не проводиться взагалі. | Розробники тестують власний код. Попередня розробка тестів. Окрема група тестування. | Розробники тестують власний код. Попередня розробка тестів. Окрема група тестування. Окрема група гарантії якості. |
| Впровадження застосунку | Неформальна процедура впровадження. | Формальна процедура впровадження. | Формальна процедура впровадження. |

Таким чином, при розробці бізнес-систем переважно використовують високоітеративні методи, при яких планування, вироблення вимог і проектування архітектури пересікаються з конструюванням, тестуванням системи та гарантією якості. Системи, від яких залежить життя людей, вимагають більш послідовних підходів, оскільки стабільність вимог – одна із умов високої надійності системи.

Вибір методу (методології) конструювання є ключовим аспектом для планування конструкторської діяльності. Такий вибір значущий для всієї конструкторської діяльності, а також необхідних умов її здійснення, визначаючи порядок відповідних операцій та рівень виконання заданих умов перед тим як почнеться конструювання або складові його дії. Наприклад, модульне тестування в ряді методів є частиною робіт, після написання відповідного функціонального коду, в той час, як ряд гнучких (agile) практик, наприклад, XP (до речі, першими почали використовувати такі методи верифікації коду), вимагають написання Unit-тестів до того, як пишеться відповідний код, що вимагає тестування.

Використовуваний підхід до конструювання впливає на можливість зменшення (в ідеалі - мінімізації) складності, готовності до змін і конструюванні з можливістю перевірки.

Планування конструкторської діяльності визначає порядок, в якому створюються компоненти та інші активи цієї галузі знань (фази діяльності), проводяться роботи по забезпеченню якості одержуваного програмного забезпечення, розподіляються завдання і відповідні ресурси, в тому числі, визначаються призначення / відображення робіт конкретним інженерам-програмістам, членам проектної групи. Все це, звичайно, відбувається, дотримуючись правил, що визначаються використовуваним методом (методологією, практиками і т.п.). Розподіл завдань має на увазі процес, що призводить до забезпечення явного зв'язку між завданням і ресурсами. У нечітко регламентованих та неформальних методах, таких, як XP, члени проектної групи самі приймають на себе відповідальність за вирішення певних завдань, а "володіння" кодом є спільним на основі співробітництва, як одного з ключових принципів роботи проектної команди.

Конкретні продукти реалізуються через методологію розробки програмного забезпечення. Методологія - це ядро теорії управління розробкою якого-небудь програмного забезпечення. Вона буде визначати, яким чином фахівець стане виконувати свою роботу. Визначення методології виконується з урахуванням розміру команди, складності і специфіки певного проекту, зрілості і стабільності процесів в компанії-роботодавця, особисті уподобання творця. Сьогодні використовується щодо методологій використовується сучасна загальна класифікація, де вони розділяються на дві групи: прогнозовані і адаптивні (раніше вони поділялися, виходячи з застосовуваних моделей життєвого циклу, на ітераційні та водоспадні). Розглянемо їх з точки зору проведення планування конструювання.

**Прогнозовані методології**

Ці методології розробки програмного забезпечення орієнтовані на детальне планування майбутнього. Завдання і ресурси відомі на всьому протязі терміну проекту. Звідси робоча команда буде важко реагувати на несподівані зміни.

План складається за складом необхідних робіт, вимог до них. Звідси зміна вимог призводить до зміни всього плану, дизайну проекту. Для прогнозованих методологій типово створення спеціального комітету, керуючого змінами, щоб у проекті враховувалися тільки найважливіші вимоги.

Більш послідовний підхід можна обирати, якщо:

* вимоги досить стабільні;
* проект відносно простий та зрозумілий;
* група розробників знайома з прикладною областю;
* проект не пов’язаний з особливим ризиком;
* важлива довготривала передбачуваність проекту;
* затрати на зміну вимог, проекту додатку і коду, скоріше за все, виявляться високими.

**Адаптивні методології**

Ці методології розробки комп'ютерного програмного забезпечення націлені на подолання очікуваного недосконалості, неповноти вимог, на постійну зміну останніх. Відповідно, зі зміною вимог буде замінюватися і команда розробників проекту.

Точний план з адаптивної методології розробляється лише на найближчий час. Плани, пов'язані з більш віддаленим від реальності подій, існують у формі декларацій про мету роботи, її результати та очікуваних витратах.

**Гнучкі методології**

Гнучкі методології розробки програмного забезпечення (англ. Agile software development) часто називаються agile-методи.

Гнучкі методології розробки програмного забезпечення - це комплекс підходів до розробки ПЗ, орієнтований на використання ітеративних розробок, динамічне формування вимог до проекту, забезпечення реалізації в результаті безперервної взаємодії всередині робочих самоорганізованих груп, складених з фахівців різного профілю.

В першу чергу, це ефективна практика трудової діяльності невеликих команд, що займаються однотипною творчою роботою. Поєднується з комбінованим (демократичним і ліберальним) методом управління.

Гнучкі методології розробки програмного комп'ютерного забезпечення спрямовані на мінімізацію ризиків шляхом приведення спільного проекту до комплексу коротких циклів (так званих ітерацій), кожен з яких максимально триває 2-3 тижні.

Ітерація тут - мініатюрний програмний проект, який включає в себе всі завдання по забезпеченню функціонального міні-приросту, а саме: планування, аналізування вимог, проектування, програмування, тестування розробки, документування. Звичайно, окремої ітерації тут недостатньо для випуску кінцевого продукту, але до кінця кожної ітерації готовий "гнучкий" програмний продукт як часткова реалізація кінцевого. Також по закінченні періоду команда обов'язково виконує переоцінку пріоритетів розробки.

Під час кожної ітерації (етапу розробки програмного забезпечення) робиться наголос на безпосередню комунікацію фахівців "лицем до лиця". Більшість команд розташовується в одному офісі. Обов'язково присутність "замовника" - повноважного представника, який пред'являє вимоги до розробки. З цією роллю справляється менеджер проекту, клієнт-замовник, бізнес-аналітик. В офісі також можуть перебувати тестувальники, технічні письменники, дизайнери інтерфейсу та ін. Основним показником тут виступає кінцевий продукт. Плюс безпосереднього спілкування фахівців у тому, що тут порівняно маленький обсяг супутньої письмовій документації.

Більш ітеративному підходу (питання вирішуються підчас роботи) краще віддавати перевагу у випадку:

* вимоги відносно незрозумілі або здається, що вони нестабільні;
* проект великим, не зовсім зрозумілий;
* група розробників незнайома з прикладною областю;
* проект пов’язаний з високим ризиком;
* довготривала передбачуваність не грає особливої ролі;
* затрати на зміну вимог, проекту додатку і коду, скоріше за все, виявляться низькими.

Суть одного популярного практичного правила зводиться до того, щоб

* заздалегідь визначити близько 80% вимог;
* передбачити час для пізнішого визначення додаткових вимог;
* виконувати по мірі роботи систематичний контроль змін, приймаючи лише найважливіші вимоги.

**Попередні умови, які необхідно виконати перед конструюванням**:

1. Чітко сформулювати проблему, яку система повинна вирішувати. Визначення проблеми – це формулювання її суті без можливих розв’язків, займає кілька сторінок і звучить, як проблема.

Це стоїть перед етапом вироблення вимог, формулюється на мові, зрозумілій користувачу й описується з користувацької точки зору. Якщо проблема не пов’язана з комп’ютерами, не слід використовувати комп’ютерну термінологію. Не маючи хорошого визначення проблеми, можна витратити зусилля на вирішення не тієї проблеми.

2. Вироблення вимог описує, що повинна робити програмна система. Ще називається аналізом вимог, специфікацією, функціональною специфікацією. Явні вимоги дозволяють гарантувати, що функціональність програми визначається користувачем, а не програмістом. Увага до вимог може допомогти звести до мінімуму зміни системи під час розробки. В результаті досліджень прийшли до висновку, що знаходження та виправлення помилки у вимогах до великих проектів на етапі проектування архітектури обходиться втричі дорожче, ніж на етапі вироблення вимог. Аналогічна помилка, виявлена при кодуванні буде дорожче у 5-10 разів, при тестуванні системи – в 10, а після випуску програми – в 10-100 разів. У менших проектах цей показник наближається до 5-10 разів. Згідно досліджень IBM при реалізації середнього проекту вимоги під час розробки змінюються приблизно на 25%.

3. Розробка архітектури, як правило, описується в єдиному документі, що називається «специфікацією архітектури» або «високорівневим проектом». Якість архітектури визначає концептуальну цілісність системи, котра, в свою чергу, визначає підсумкову якість системи. Архітектура дозволяє розбити роботу на частини для груп чи окремих програмістів. Вона визначає:

* основні класи додатку та механізми їх взаємодії з іншими класами. Якщо система досить велика, то слід описувати організацію класів у підсистеми. Звертається увага на альтернативні варіанти організації та обґрунтування того, чому саме було обрано цей варіант;
* організацію даних у вигляді опису форматів файлів, таблиць, БД (з альтернативами та обґрунтуванням). Ця інформація допоможе при конструюванні та супроводженні програмної системи, підказуючи, чим керувалися розробники;
* специфічні бізнес-правила та їх вплив на проект (наприклад, правило, що дані в системі оновлюються кожні 30с);
* інтерфейс користувача (описує головні елементи формату Web-сторінок, GUI, інтерфейсу командного рядка);
* план управління обмеженими ресурсами (з’єднання з БД, потоки, дескриптори). При умові обмеження об’єму пам’яті архітектура повинна визначати спосіб керування пам’яттю. Архітектура повинна включати оцінку ресурсів, що будуть використовуватись при номінальному та екстремальному навантаженні;
* підхід до безпеки на рівні проекту додатка й на рівні коду (методики обробки буферів та ненадійних даних – даних, введених користувачами, конфігураційних даних, cookie-файлів та інших даних із зовнішніх інтерфейсів; підходи до шифрування, рівня докладності повідомлень про помилки, захист секретних даних, що знаходяться в пам’яті та ін.);
* масштабованість – здатність системи адаптуватись до росту вимог (реагування системи на збільшення кількості користувачів, серверів, мережевих вузлів, записів до БД, транзакцій тощо);
* взаємодію з іншими системами (при наявності спільних даних чи ресурсів з іншими програмами чи пристроями);
* інтернаціоналізацію/локалізацію програмної системи (підтримка регіональних стандартів та мов). Особлива увага цьому питанню приділяється при розробці архітектури інтерактивної системи, що включає підказки, індикатори стану, повідомлення про помилки, допоміжні повідомлення та ін.;
* введення-виведення (схема зчитування даних, рівень, на якому будуть визначатись помилки введення-виведення: на рівні полів, записів, потоків даних чи файлів);
* обробку помилок;
* відмовостійкість;
* можливість реалізації архітектури (чи зможе система досягнути заданих показників продуктивності, працювати при обмеженості ресурсів). Архітектура повинна підтверджувати, що система може бути технічно реалізована;
* надлишкову функціональність (явно вказувати, чи можуть програмісти її реалізувати у своїх блоках програми, чи повинні створювати найпростішу працездатну систему);
* купівлю чи створення. Якщо архітектура не передбачає використання готових компонентів, вона повинна пояснювати, в яких аспектах розроблювані компоненти будуть краще за готові бібліотеки та компоненти);
* повторне використання (як ресурси, що будуть повторно використані, адаптуються у програмну систему);
* стратегію змін (архітектура повинна показувати, що розглядалися можливі покращення і що реалізація можливих покращень виявиться найбільш простою);
* якщо проект розвивається без проблем, то на вироблення вимог, архітектуру та попереднє планування витрачається 10-20% зусиль та 20-30% часу.