ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Кафедра прикладної математики і інформатики

МЕТОДИЧНІ ВКАЗІВКИ І ЗАВДАННЯ ДО ВИКОНАННЯ КУРСОВОГО ПРОЕКТУ З ДИСЦИПЛІНИ "КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ "

(для студентів, що навчаються за спеціальністю 121 Інженерія програмного забезпечення)

Покровськ, 2019

УДК 004.415.2

Методичні вказівки і завдання до виконання курсового проекту з дисципліни "Конструювання програмного забезпечення" (для студентів, що навчаються за спеціальністю 121 Інженерія програмного забезпечення). Укл. Н.С.Костюкова. – Покровськ: ДВНЗ «ДонНТУ», 2019. – 14 с.

Укладач: *Костюкова Наталя Стефанівна, к.т.н., доцент кафедри прикладної математики та інформатики*

Рецензент: *Башков Є.О., д.т.н., професор*

Відповідальний за випуск: *О.А.Дмитрієва, д.т.н., професор, завідувач кафедри прикладної математики та інформатики*

Затверджено на засіданні навчально- видавничої ради ДонНТУ, протокол № від .

Розглянуто на засіданні кафедри прикладної математики і інформатики, протокол № 8 від 24.01.2019.

Покровськ, ДВНЗ «Донецький національний технічний університет», 2019

ВСТУП

Предметом вивчення навчальної дисципліни «Конструювання програмного забезпечення» є процес створення працюючої функціональної програмної системи за допомогою кодування, верифікації, модульного тестування, інтеграційного тестування та відладки.

Завданнями дисципліни є:

• вивчити основи моделювання, моделі конструювання, типи моделей, планування конструювання, мови конструювання, інтеграцію, шаблони проектування;

• оволодіти основами конструювання ПЗ; застосовуванням та створенням компонентів багаторазового використання.

У результаті вивчення навчальної дисципліни “Конструювання програмного забезпечення” студент повинен знати: основи моделювання, моделі конструювання, типи моделей, планування конструювання, мови конструювання, інтеграцію, шаблони проектування; вміти: конструювати програмне забезпечення, застосовувати та створювати компоненти багаторазового використання.

В ході виконання передбаченого навчальним планом курсового проекту студент повинен виконати таке **завдання**:

створити проект програмної системи, що включає план розробки програми, підсистему, створену за методикою екстремального програмування, компонентну модель системи, елементи проекта за методикою SCRUM: беклог проекту, беклоги спринтів, діаграма згоряння задач.

На початку семестру складається завдання на курсовий проект, яке підписується студентом і керівником і затверджується завідувачем кафедри. При захисті курсового проекту завдання є основним документом, який дає можливість оцінити повноту виконання роботи студентом.

Приклад оформлення завдання наведено у додатку А.

1. ВИКОНАННЯ РОЗДІЛІВ КУРСОВОГО ПРОЕКТУ

1.1. **Планування організації робіт над проектом програм**

Література: [1, 5, 10, 11]

1.1.1. Вказівки до виконання завдання

В результаті виконання цього завдання студент має набути практичних навичок в застосуванні методів мережевого планування розробки великих програмних систем в задані терміни і з оцінкою необхідних ресурсів.

Результатом виконання завдання є раціональний мережевий графік реалізації проекту програмного комплексу відповідно до варіанта.

Для цього потрібно проаналізувати принципи традиційних стратегій проектування "зверху-вниз", "знизу-вгору" або "вертикальне шарування" і сформувати свій підхід до проектування програми, що поєднує переваги традиційних стратегій, специфіку розв'язуваної задачі, індивідуальний досвід і організаційні стереотипи. При плануванні порядку розробки програмних модулів використовувати ієрархічний, операційний або комбінований підходи.

Планування порядку розробки програмних модулів слід почати зі складання діаграми майбутніх робіт. Діаграма будується з урахуванням обраного підходу до проектування комплексу, структура якого визначена схемою складу розкладання.

Для кожної роботи, представленої на діаграмі, потрібно вибрати наступні вихідні дані: тривалість роботи, інтенсивність розробки модуля і т.д. Вихідні дані визначити за допомогою експертних оцінок передбачуваних витрат часу програміста (ів) на розробку і тестування кожного модуля окремо і необхідних для цього обчислювальних ресурсів. Після цього слід виконати розрахунок параметрів мережного графіка. Далі необхідно зобразити мережевий графік, враховуючи ранні початки кожної роботи, із зазначенням

наявних резервів часу; визначити критичні роботи і намалювати графік розподілу ресурсів.

З урахуванням конкретної ситуації слід побудувати субоптимальний мережевий графік, перерозподіляючи роботи в межах наявних резервів часу, і відповідну йому діаграму розподілу людських ресурсів.

Базовим поняттям мережевого планування є робота - витрати, які пов'язані з проектуванням, кодуванням і тестуванням одного модуля.

Вихідними даними для планування є:

• тривалість кожної k-ой роботи (Tk-витрати на проектування, кодування і тестування модуля, дні),

• інтенсивність розробки модуля (Qk, людина / день),

• кошти на виконання робіт (Ck, грн.) І т.д.

Ці дані готуються досвідченими програмістами на основі особистого досвіду, статистичних даних та експертних оцінок.

Метою планування є складання мережного графіка.

В першу чергу складається діаграма висхідного або спадного проектування. Для кожної роботи задається вага - номер рівня в дереві ієрархії (знизу - при висхідному проектуванні, зверху - при низхідному).

Після цього роботи сортуються за зростанням ваги і для кожної роботи знаходиться безліч які безпосередньо передують їй робіт і безліч безпосередньо наступних робіт.

Потім визначається найбільш ранній термін закінчення кожної роботи за формулою:

$$T'_k = \max_{i \in \omega'_k} (T'_i + \tau_k).$$

Час завершення всього комплексу робіт визначається наступним чином:

$$T = \max_k T'_k;$$

Після цього визначається пізній термін закінчення кожної роботи:

$$T_k'' = \min_{i \in \omega_k^*} (T_i'' - \tau_i),$$

де $T_N'' = T$.

Дана характеристика визначається від останньої роботи (в ранжованому списку) до першої.

Потім обчислюються резерви часу для кожної роботи:

$$\Delta\tau_k = T_k'' - T_k'.$$

Резерви часу є важливою характеристикою, яка б показала, на який термін можна розтягнути виконання роботи, не збільшуючи час виконання всього комплексу робіт. Роботи з нульовим резервом називаються критичними, їм повинна приділятися більше уваги, ніж іншим.

Після цього обчислюються ранні початку кожної роботи:

$$T_k^0 = T_k' - \tau_k.$$

Після цього складають мережевий графік і діаграму розподілу людських ресурсів.

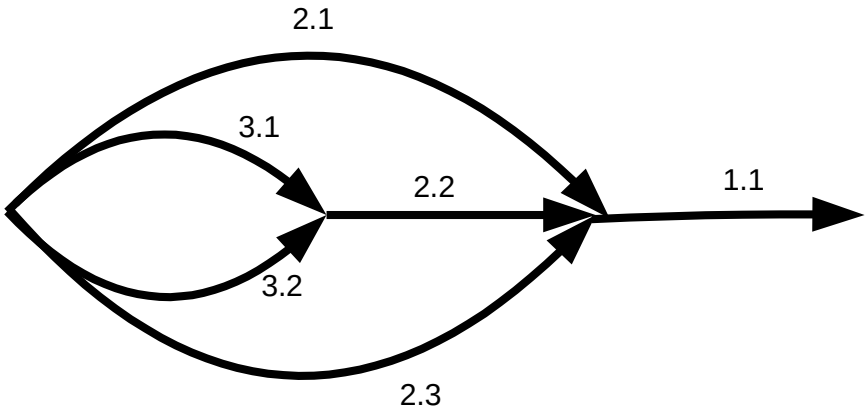
З метою оптимального розподілу ресурсів перерозподіляють роботи в межах наявних резервів часу.

1.1.2. Приклад виконання завдання

Як **приклад** виконаємо розрахунок мережевого графіка для **розробки гіпертекстової навчальної системи по одному з розділів програмування.**

Схема ієрархії гіпертекстової навчальної системи може виглядати так:

Діаграма висхідного проектування:



Сформуємо вихідні дані для розробки мережевого графіка:

Робота	Вага	Тривалість T_k	Інтенсивність Q_k
1.1	3	1	1
2.1	1	2	1
2.2	2	1	1
2.3	1	4	1
3.1	1	4	1
3.2	1	4	1

Виконаємо ранжування робіт за вагою:

Робота	Вага	Тривалість T_k	Інтенсивність Q_k
2.1	1	2	1
2.3	1	4	1
3.1	1	4	1
3.2	1	4	1
2.2	2	1	1
1.1	3	1	1

Далі сформуємо множину безпосередньо попередніх (ω_k') і безпосередньо наступних (ω_k'') робіт:

Робота	Вага	ω_k'	ω_k''
2.1	1	–	1.1
2.3	1	–	1.1
3.1	1	–	2.2

3.2	1	–	2.2
2.2	2	3.1, 3.2	1.1
1.1	3	2.1, 2.2, 2.3	–

Визначимо найбільш ранній термін закінчення кожної роботи:

Работа	ω_k'	T_k	T_k’
2.1	–	2	Max(0)+2=2
2.3	–	4	Max(0)+4=4
3.1	–	4	Max(0)+4=4
3.2	–	4	Max(0)+4=4
2.2	3.1, 3.2	1	Max(4,4)+1=5
1.1	2.1, 2.2, 2.3	1	Max(2,5,4)+1=6

Час завершення проекту T=max (2,4,4,4,5,6)=6.

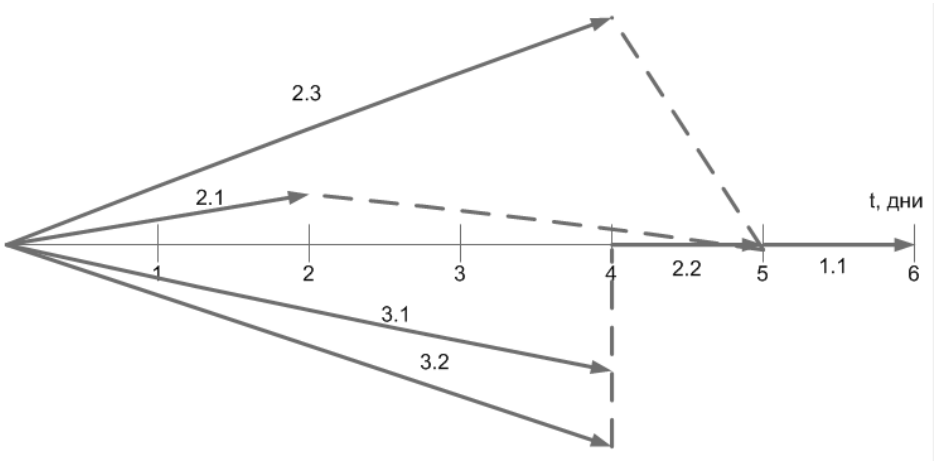
Розрахуємо пізній термін закінчення кожної роботи в такій послідовності: 1.1, 2.2, 3.2, 3.1, 2.3, 2.1 (від останньої роботи до першої):

Робота	ω_k''	T_k	T_k’’
2.1	1.1	2	min(6)-1=5
2.3	1.1	4	min(6)-1=5
3.1	2.2	4	min(5)-1=4
3.2	2.2	4	min(5)-1=4
2.2	1.1	1	min(6)-1=5
1.1	–	1	6

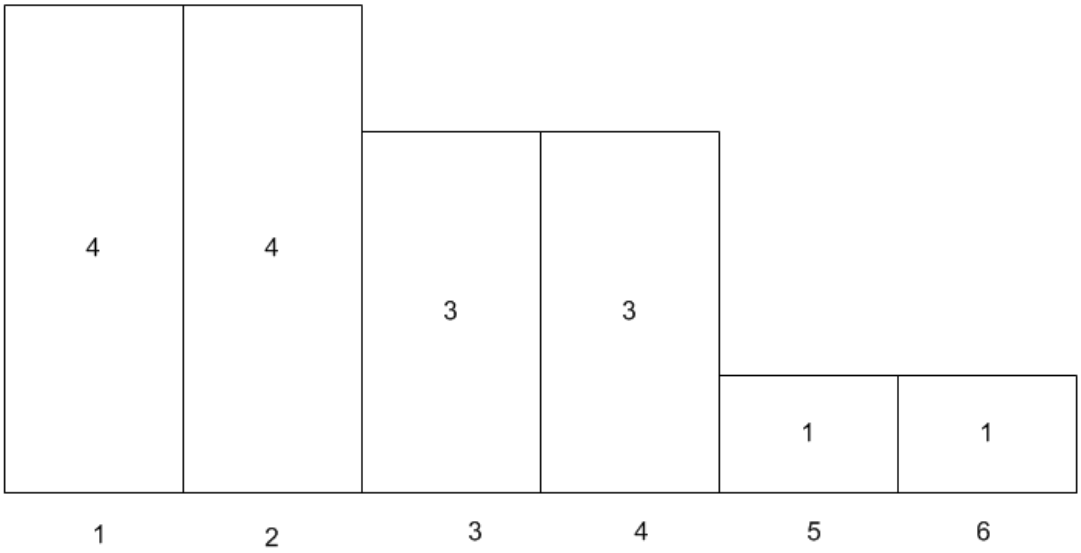
Визначимо резерви і ранні початки робіт:

Робота	T_k	T_k’	T_k’’	Резерв	Ранній початок
2.1	2	2	5	3	0
2.3	4	4	5	1	0
3.1	4	4	4	0	0
3.2	4	4	4	0	0
2.2	1	5	5	0	4
1.1	1	6	6	0	5

Складемо мережевий графік робіт:



Розподіл ресурсів під час виконання робіт:



При необхідності початок робіт 2.1 і 2.3 можна зсунути в межах резервів, змінивши розподіл ресурсів на більш зручний.

1.1.3 Зміст розділу

В результаті виконання завдання за даним розділом студент має представити для оцінки наступні елементи:

1. Схема ієрархії програмного комплексу. Опис обраної стратегії (підходу) проектування комплексу.

2. Вихідні дані, первісна діаграма робіт, відповідна застосовуваної стратегії проектування.

3. Розрахунок параметрів мережного графіка в табличній і графічній формі.

4. Субоптимальний графік робіт, розподіл ресурсів, критичні роботи, загальний термін виконання проекту.

5. Рекомендації щодо можливої зміни початку робіт.

1.1.4. Контрольні питання

1. У чому перевага мережевого планування при розробці великих програмних систем?

2. Що є вихідними даними при мережевому плануванні?

3. Для чого потрібен мережевий графік робіт і як він складається для програмних комплексів?

4. Яким способом ранжуються роботи?

5. Як визначається критичний шлях на мережевому графіку і що він означає?

6. Завдяки чому можлива оптимізація графіка виконання робіт і необхідних ресурсів на реалізацію проекту?

1.2. **Розробка програми з використанням методики екстремального програмування**

Література: [2, 3, 13, 14]

1.2.1. Вказівки до виконання завдання

В результаті виконання цього завдання студент має набути практичних навичок у застосуванні методики екстремального програмування при вирішенні різних задач. Завдання полягає у створенні програми для вирішення задачі, вказаної у варіанті, і набору тестів для застосування екстремального програмування.

При виконанні завдання слід написати програму, яка вирішує задачу на штатному наборі даних, протестувати її, після чого виділити характеристики вхідних даних, для яких програма не дає рішення, і внести відповідні зміни до програми.

Екстремальне програмування (від англ. Extreme Programming, або XP) - це спрощена технологія створення програмного забезпечення в умовах нечітких та постійно змінних вимог для невеликих колективів розробників.

Основними принципами XP є:

- ітеративність;
- прості робочі рішення;
- розробка невеликими групами чи парами, інтенсивна розробка;
- зворотній зв'язок із замовником;
- зустрічі для обговорення проблем, що виникають у процесі розробки;
- ризик під час розробки.

Ітеративність передбачає, що розробка здійснюється короткими ітераціями, від 1-2 тижнів до місяця, за умови активної комунікації із замовником.

Прості робочі рішення — це реалізація мінімального набору головних функцій системи на кожній ітерації.

Розробка невеликими групами чи парами, інтенсивна розробка передбачає, що у розробці беруть участь не більше десяти осіб; у випадку парного програмування два спеціаліста створюють код на одному робочому місці. При розробці використовується open-простір для роботи з метою надання можливості підтримувати комунікацію усіх розробників.

Зустрічі призначені для обговорення проблем, що виникають у процесі розробки, та шляхів усунення цих проблем для посилення концентрації команди. Зустрічі тривають недовго, не вимагаючи від робітників витрат за часом і обговорення не цікавих їм тем.

Головною метою екстремального програмування є зменшення вартості неочікуваних змін. Традиційно вимоги до системи визначаються на початку роботи над проектом, і часто виправляються пізніше. Це означає, що вартість проекту через зміни буде більшою за заплановану (традиційна особливість для програмного забезпечення, що проектується).

XP використовується для зменшення вартості внесення змін у програму. Для цього використовується командна робота з тісними зв'язками усередині команди та із замовником, розробка найбільш простих працюючих рішень, гнучке адаптивне планування, оперативний зворотний зв'язок (шляхом модульного та функціонального тестування).

Основними принципами XP є розробка невеликими ітераціями на підставі порції вимог замовника (так званих користувальницьких історій), написання функціональних тестів до написання програмного коду, постійне спілкування і постійний рефакторинг коду.

Основними практиками XP є:

планування процесу;
часті релізи;
метафора системи;
проста архітектура;
тестування;
рефакторинг;
парне програмування;
колективне володіння кодом;
часта інтеграція;
40-годинний робочий тиждень;
стандарти кодування;
тісна взаємодія із замовником.

Екстремальне програмування виникло як еволюційний метод розробки ПЗ "знизу вгору". Цей підхід є прикладом так званого методу "живої" розробки (Agile Development Method).

Основні принципи "живої" розробки ПЗ зафіксовані в маніфесті "живої" розробки [15], який з'явився у 2000 році.

Люди, які беруть участь у проекті, та їхнє спілкування більш важливі, ніж процеси та інструменти.

Працююча програма більш важлива, ніж вичерпна документація.

Співпраця із замовником більш важлива, ніж обговорення деталей контракту.

Відпрацювання змін більш важливе, ніж дотримання планів.

"Живі" методи з'явилися як протест проти надмірної бюрократизації розробки ПЗ. Велика частина таких робіт та документів не має прямого відношення до розробки ПЗ та забезпечення його якості, а призначена для дотримання формальних пунктів контрактів на розробку, отримання та підтвердження сертифікатів на відповідність різним стандартам. "Живі" методи дозволяють велику частину зусиль розробників зосередити власне на завданнях розробки і задоволення реальних потреб користувачів.

Однак, XP має свою схему процесу розробки, наведену на рис. 12.

За твердженням авторів XP, ця методика являє собою не стільки слідування якимось загальним схемами дій, скільки застосування комбінації наступних технік. При цьому кожна техніка важлива, і без її використання розробка вважається йде не по XP.

Дванадцять основних прийомів екстремального програмування (за першого видання книги Extreme programming explained) можуть бути об'єднані в чотири групи:

Короткий цикл зворотного зв'язку (Fine scale feedback)

розробка через тестування (Test driven development);

гра в планування (Planning game);

замовник завжди поруч (Whole team, Onsite customer);

парне програмування (Pair programming).

Безперервний, а не пакетний процес

безперервна інтеграція (Continuous Integration);

рефакторинг (Design Improvement, Refactor);

часті невеликі релізи (Small Releases).

Розуміння, поділюване всіма

простота (Simple design);

метафора системи (System metaphor);

колективне володіння кодом (Collective code ownership) або обраними шаблонами проектування (Collective patterns ownership);

стандарт кодування (Coding standard or Coding conventions).

Соціальна захищеність програміста (Programmer welfare):

40-годинний робочий тиждень (Sustainable pace, Forty hour week).

Тестування

У XP особлива увага приділяється двом різновидам тестування:

тестування модулів (unit testing);

приймальне тестування (acceptance testing).

Розробник не може бути впевнений у правильності написаного ним коду до тих пір, поки не спрацюють абсолютно всі тести модулів розроблюваної ним системи. Тести модулів дозволяють розробникам переконатися в тому, що їхній код працює коректно. Вони також допомагають іншим розробникам зрозуміти, навіщо потрібен той чи інший фрагмент коду і як він функціонує. Тести модулів також дозволяють розробнику без будь-яких побоювань виконувати рефакторінг (refactoring).

Приймальні тести дозволяють переконатися в тому, що система дійсно має заявлені можливості. Крім того, приймальні тести дозволяють перевірити коректність функціонування розроблюваного продукту.

Для XP більш пріоритетним є підхід називаний TDD (Test Driven Development), спочатку пишеться тест, який не проходить, потім пишеться код, щоб тест пройшов, а вже після робиться рефакторинг коду.

Гра в планування.

Основна мета гри в планування - швидко сформувати приблизний план роботи і постійно оновлювати його в міру того, як умови завдання стають все більш чіткими. Артефактами гри в планування є набір паперових карток, на яких записані побажання замовника (customer stories), і приблизний план роботи з випуску наступних однієї або декількох невеликих версій продукту. Критичним чинником, завдяки якому такий стиль планування виявляється ефективним, є те, що в даному випадку замовник відповідає за прийняття бізнес-рішень, а команда розробників відповідає за прийняття технічних рішень. Якщо не виконується це правило, весь процес розпадається на частини.

Замовник завжди поруч.

"Замовник" в XP - це не той, хто оплачує рахунки, а той, хто насправді використовує систему. XP стверджує, що замовник повинен бути весь час на зв'язку і доступний для запитань.

Парне програмування.

Парне програмування припускає, що весь код створюється парами програмістів, які працюють за одним комп'ютером. Один з них працює безпосередньо з текстом програми, другий переглядає його роботу і стежить за загальною картиною того, що відбувається. При необхідності клавіатура вільно передається від одного до іншого. Протягом роботи над проектом пари не фіксовані: рекомендується їх перемішувати, щоб кожен програміст у команді мав чітке уявлення про всю систему. Таким чином, парне програмування посилює взаємодію всередині команди.

Безперервна інтеграція.

Якщо ви будете виконувати інтеграцію розроблюваної системи досить часто, ви зможете уникнути більшої частини пов'язаних з цим проблем. У традиційних методиках інтеграція, як правило, виконується в самому кінці роботи над продуктом, коли вважається, що всі складові частини системи, яка розробляється, повністю готові. У XP інтеграція коду всієї системи виконується кілька разів на день, після того, як розробники переконалися в тому, що всі тести модулів коректно спрацьовують.

Рефакторинг.

Рефакторинг (refactoring) - це методика поліпшення коду, без зміни його функціональності. XP має на увазі, що одного разу написаний код в процесі роботи над проектом майже напевно буде неодноразово перероблений. Розробники XP безжально переробляють написаний раніше код для того, щоб поліпшити його. Цей процес називається рефакторингом. Відсутність тестового покриття провокує відмову від рефакторингу, у зв'язку з побоюванням поламати систему, що призводить до поступової деградації коду.

Часті невеликі релізи.

Версії (releases) продукту повинні надходити в експлуатацію як можна частіше. Робота над кожною версією повинна займати якомога менше часу. При цьому кожна версія повинна бути досить осмисленою з точки зору корисності для бізнесу.

Чим раніше ми випустимо першу робочу версію продукту, тим раніше замовник почне отримувати за рахунок неї додатковий прибуток. Слід пам'ятати, що гроші, зароблені сьогодні, коштують дорожче, ніж гроші, зароблені завтра. Чим раніше замовник приступить до експлуатації продукту, тим раніше розробники отримають від нього інформацію про те, що відповідає вимогам замовника, а що - ні. Ця інформація може виявитися надзвичайно корисною при плануванні наступного випуску.

Простота дизайну.

XP виходить з того, що в процесі роботи умови задачі можуть неодноразово змінитися, а значить, розроблюваний продукт слід проектувати завчасно цілком і повністю. Якщо на самому початку роботи ви намагаєтеся

від початку і до кінця детально спроектувати систему, ви марно витрачаєте час. XP припускає, що проектування - це настільки важливий процес, що його необхідно виконувати постійно на протязі усього часу роботи над проектом. Проектування має виконуватися невеликими етапами, з урахуванням вимог, які постійно змінюються. У кожний момент часу потрібно намагатися використовувати найбільш простий дизайн, який підходить для рішення поточної задачі. При цьому доцільно міняти його в міру того, як умови задачі змінюються.

Метафора системи.

Архітектура - це деяке уявлення про компоненти системи і про те, як вони взаємопов'язані між собою. Розробники використовують архітектуру для того, щоб зрозуміти, в якому місці системи додається деяка нова функціональність і з чим буде взаємодіяти деякий новий компонент.

Метафора системи (system metaphor) - це аналог того, що в більшості методик називається архітектурою. Метафора системи дає команді уявлення про те, яким чином програма працює в даний час, в яких місцях додаються нові компоненти і яку форму вони повинні прийняти.

Підібравши хорошу метафору, ви полегшуєте команді розуміння того, яким чином улаштована ваша система. Іноді зробити це не просто.

Стандарти кодування.

Усі члени команди в ході роботи повинні дотримуватися вимоги загальних стандартів кодування. Завдяки цьому:

Члени команди не витрачають час на дурні суперечки про речі, які фактично ніяк не впливають на швидкість роботи над проектом.

Забезпечується ефективне виконання інших практик.

Якщо в команді не використовуються єдині стандарти кодування, розробникам стає складніше виконувати рефакторинг; при зміні партнерів у парах виникає більше ускладнень; загалом і в цілому, просування проекту ускладнюється. У рамках XP необхідно добитися того, щоб було складно зрозуміти, хто є автором тієї чи іншої ділянки коду, - вся команда працює уніфіковано, як одна людина. Команда повинна сформувати набір правил, а потім кожен член команди повинен слідувати цим правилам у процесі кодування. Перелік правил не повинен бути вичерпним або занадто об'ємним. Завдання полягає в тому, щоб сформулювати загальні вказівки, завдяки яким код стане зрозумілим для кожного з членів команди. Стандарт кодування спочатку повинен бути простим, потім він буде еволюціонувати в міру того, як команда набуває досвіду. Ви не повинні витрачати на попередню розробку стандарту занадто багато часу.

Колективне володіння.

Колективне володіння означає, що кожен член команди несе відповідальність за весь вихідний код. Таким чином, кожен має право вносити зміни до будь-якої ділянки програми. Парне програмування підтримує цю практику: працюючи в різних парах, всі програмісти знайомляться з усіма частинами коду системи. Важлива перевага колективного володіння кодом - у тому, що воно прискорює процес розробки, оскільки при появі помилки її може усунути будь-який програміст.

Даючи кожному програмісту право змінювати код, ми отримуємо ризик появи помилок, що вносяться програмістами, які вважають, що знають

що роблять, але не розглядають деякі залежності. Добре визначені UNIT-тести вирішують цю проблему: якщо не розглянуті залежності породжують помилки, то наступний запуск UNIT-тестів буде невдалим.

Як видно із застосовуваних технік, XP розраховане на використання у рамках невеликих команд (не більше 10 програмістів), що підкреслюється авторами цієї методики. Більший розмір команди руйнує необхідну для успіху простоту комунікації і робить неможливим застосування багатьох перелічених прийомів.

Перевагами XP, якщо його вдається застосувати, є більша гнучкість, можливість швидко й акуратно вносити зміни до ПЗ у відповідь на зміни вимог і окремі побажання замовників, у результаті висока якість отриманого коду і відсутність необхідності переконувати замовників у тому, що результат відповідає їхнім очікуванням.

Недоліками цього підходу є нездійсненність у такому стилі досить великих і складних проектів, неможливість планувати терміни і трудомісткість проекту на досить довгу перспективу і чітко передбачити результати тривалого проекту в термінах співвідношення якості результату й витрат часу і ресурсів. Також можна відзначити непристосованість XP для тих випадків, в яких можливі рішення не знаходяться відразу на основі раніше отриманого досвіду, а вимагають проведення попередніх досліджень.

1.2.3 Зміст розділу:

1. Початковий варіант програми і опис вхідних даних, для яких вона виконується коректно.
2. Остаточний варіант програми і опис вхідних даних.
3. Результати роботи програми (екранні форми).

1.2.4 Контрольні питання

1. Що характерно для методики екстремального програмування?
2. У чому переваги методики екстремального програмування?
3. У чому недоліки методики екстремального програмування?

1.3. **Розробка програми з використанням методики компонентне програмування**

Література: [5, 6, 13, 14]

При виконанні цього завдання студент має набути практичних навичок в застосуванні методики компонентного програмування. Для цього у середовищі візуального програмування слід створити програму для вирішення зазначеної в варіанті завдання і компонент з відповідним методом. Інтегрувати компонент в бібліотеку середовища розробки. Спершу потрібно аписати програму для вирішення завдання за варіантом, для інтерфейсу з користувачем використовувати компоненти, що надаються обраної середовищем розробки або Інтернет. Потім на базі компонента для введення вихідних даних створити свій компонент з методом, що надає рішення задачі. Далі слід інтегрувати розроблений компонент в бібліотеку середовища розробки, і створити програму, яка ілюструє роботу з компонентом, включеним в бібліотеку.

Зміст розділу:

1. Текст програми для вирішення задачі з виділенням компонентів середовища розробки.
2. Опис розробленого компонента і послідовності дій по його інтеграції в бібліотеку.
3. Текст програми, що ілюструє роботу з компонентом, включеним в бібліотеку.

Контрольні питання

1. У чому суть компонентного програмування?

2. Які сучасні середовища програмування підтримують компонентний підхід?

3. Яка типова структура компонента?

1.4.Розробка проекту програми за методикою SCRUM

Література: [1, 2, 7, 8, 13, 14]

При виконанні завдання студент має набути практичних навичок в застосуванні методики Scrum. Для цього потрібно сформувати опис проекту по розробці програми відповідно до варіанта.

Зміст розділу:

1. Беклог проекту.
2. Беклогі спринтів.
3. Діаграма згоряння завдань.

Контрольні питання

1. У чому суть методики Scrum?
2. Що таке спринт, беклог?
3. Для чого використовується діаграма згоряння завдань?
4. У чому суть покеру планування?

2. ВАРІАНТИ ЗАВДАНЬ

1 Визначити число днів, що залишилися до кінця поточного місяця від заданої дати

2 Визначити номер кварталу, до якого належить задана дата (1 квартал- січень-березень, 2 квартал- квітень-червень, 3 квартал- липень-сентабря, 4 квартал- жовтень-грудень)

3 Визначити дату дня, що передує заданому

4 Визначити дату дня, віддаленого від заданого на 1 тиждень

5 Визначити дату початку наступного тижня (відомі поточна дата і день тижня)

6 Визначити дату найближчого повного місяця, їли відома дата молодика (повний місяць настає через 14 днів після молодика)

7 Визначити дату виходу людини з відпустки, якщо відома дата початку відпустки і його тривалість в днях

8 Визначити число днів в заданому році

9 Визначити дату, віддалену від заданої на задану кількість тижнів

10 Визначити час року, до якого належить введена дата

11 Визначити дату кінця поточного тижня (відомі поточна дата і день тижня)

12 Визначити номер тижні від початку місяця для заданої дати

13 Визначити, на яку саму пізню дату можна придбати квиток на поїзд в заданий день (продаж квитків починається за 45 діб до дати відправлення)

14 Визначити дату дня, наступного за заданим

15 Визначити число днів в заданому місяці заданого року

16 Визначити дату найближчого новонолунія, їли відома дата повного місяця (молодик настає через 14 днів після повного місяця)

17 Визначити дату, віддалену від заданої на задане число днів

18 Визначити дату, яка раніше заданої на 1 тиждень

19 Визначити число днів в поточному місяці

20 Визначити дату початку поточного тижня, якщо відома поточна дата і день тижня

3 ПЕРЕЛІК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

Базова:

1. Погорілий С.Д. Програмне конструювання. За редакцією академіка АПН України Третяка О.В. ВПЦ Київський університет. 2005.

2. Соммервилл И. Инженерия програмного обеспечения. М.: Издательский дом “Вильямс”, 2002 г.– 624 с.

3. Брауде Эрик Дж. Технология разработки программного обеспечения. М.: Computer Science, 2004. – с. 655.

4. Андон А. И., Яшунин А. Е., Резниченко В. А. Логические модели интеллектуальных информационных систем. - К.: Наук. думка, 1999. -320 с.

5. Иванова Г.С. Технология программирования. М.: МГТУ имени Н.Э. Баумана, 2002 – 241 с.

6. Крылов Е.В., Острейковский В.А., Типикин Н.Г.Техника разработки программ: В 2-х книгах. Учебник, Кн.2: Технология, надежность и качество программного обеспечения. М.: Высшая школа, 2007 г.

7. Терехов А.Н. Технология программирования. Учебное пособие. Интернет-Университет Информационных технологий, 2012г.

Додаткова:

8. Гради Буч. Объектно-ориентированное проектирование с примерами применения: пер. с англ. - М.: Конкорд, 1992.

Додаткова:

9. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приёмы объектно-ориентированного проектирования. Патерны проектирования - СПб: Питер, 2001. — 368 с.

10. Вендров А.М. Проектирование программного обеспечения экономических информационных систем. М.: Финансы и статистика, 2002 г. - 352 с.

11. Липаев В.В. Программная инженерия: методологические основы. М.:ТЭИС, 2006 г. – 608 с.
12. Кауфман В.Ш. Языки программирования. Концепции и принципы. М.: Радио и связь, 1993.

Інформаційні ресурси

13. Планета информатики: открытый учебник по компьютерной науке и информационным технологиям: http://inf1.info/informatics
14. Портал знань: http://www.znannya.org/

Додаток А

Приклад оформлення завдання на курсовий проект

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

ЗАТВЕРДЖУЮ
зав. кафедрою ПМІ, проф.., д.т.н.
_____О.А.Дмитрієва
«_____»_____2018 р.

ЗАВДАННЯ
на курсовий проект з дисципліни
«Конструювання програмного забезпечення»
студент _____-
факультету комп'ютерних наук і технологій
групи ПІ-15
Тема роботи: конструювання програмної системи

Покровськ 2018

1. Дата видачі завдання: 5 лютого 2018 року

2. ЗАВДАННЯ:

Створити проект програмної системи, що включає план розробки програми, підсистему, створену за методикою екстремального програмування, компонентну модель системи, елементи проекта за методикою SCRUM: беклог проекту, беклоги спринтів, діаграма згоряння задач.

3. Термін захисту роботи: 26.05.2018

4. Графік виконання курсової роботи

№ з.п	Етап	Термін виконання (номер тижня)	Примітка про виконання
1	Вивчення принципів планування робіт над проектом	2	
2	Створення субоптимального графіку розробки	4	
3	Вивчення принципів екстремального програмування	6	
4	Розробка програмної підсистеми за принципами екстремального програмування	8	
5	Вивчення принципів компонентної моделі	10	
6	Створення компонентної моделі системи	12	
7	Створення проекту за методикою SCRUM	14	
7	Оформлення пояснювальної записки	15	
8	Захист роботи	16	

Студент _____ (В. І. Іванов)
Керівник роботи _____ (Н.С.Костюкова)

ЗМІСТ