ДОМАШНЯ РОБОТА
З ДИСЦИПЛІНИ
«Аналіз вимог до програмного забезпечення»

ВСТУП

Для виконання домашньої роботи необхідно розбитися на групи 3-4 особи, обрати тему проекту та здобути, а потім показати навички командної роботи в TFS з використанням гнучкої методології розробки програмного продукту.

Домашня робота спрямована на:

– перевірку теоретичних концепцій дисципліни "Основи програмної інженерії" та «Аналіз вимог до програмного забезпечення» шляхом експериментів;

– поглиблення знань про основи роботи в команді, стандартів кодування, роботу з системою контроля версій, системою відстеження дефектів, аналізу коду та інструментів профілювання;

– надбання практичних навичок в проектуванні, написанні та аналізу програм, взаємодії та роботі в команді.

Домашня робота містить 4 контрольних завдання. Усі роботи повинні бути виконані з використанням мови програмування C#.

Для виконання завдань на робочу станцію студента повинно бути встановлено наступне програмне забезпечення:

- Microsoft Visual Studio
- Microsoft Team Foundation Build
- Microsoft Office

Домашня робота містить наступні завдання:

1.1 Вивчення системи контролю версій в Team Foundation Server.

1.2 Дослідження роботи з робочими елементами, сценаріями, задачами й помилками.

1.3 Робота із запитами до робочих елементів, створення збірки.

1.4 Аналіз стану проекту за допомогою звітів, аналіз коду, профілювання застосування.

До захисту домашньої роботи допускаються студенти, які працювали над проектом, мають звіт, в якому відображено виконання кожного завдання домашньої роботи та які впевнено орієнтуються в своєму проекті та мають відповідні навики роботи в TFS.

Завдання1. Вивчення системи контролю версій в Team Foundation Server

Мета- отримати базові знання з систем контролю версій і отримати навички, що необхідні кожному розробнику програмного забезпечення у повсякденній роботі. Дізнатися, як за допомогою команд Team Foundation Server керувати версіями програмного продукту.

Задачі

1. Вивчити основи систем контролю версії .
2. Вивчити як зв'язати локальну папку з серверною.
3. Вивчити як додавати, видаляти, реєструвати, витягувати та завантажувати файли користуючись системою контролю версії Team Foundation Server.

Теорія

Система контролю версій (контролю ревізій) дозволяє відстежувати модифікації файлів з пливом часу.

Використання систем контролю версій важливе тому що дозволяє :

1. Виконувати резервне копіювання та відновлення;
2. Синхронізувати дані;
3. Короткострокове скасування;
4. Довгострокове скасування;
5. Відстежувати зміни;
6. Відстежувати власність;
7. Ізолювати;
8. Розгалужувати та зливати.

Більшість систем контролю версій використовують ці базові принципи, хоча терміни можуть бути різними.

Репозиторій– це база даних що містить файли та папки.

Сервер – це обчислювальна машина що містить репозиторій.

Клієнт – це обчислювальна машина що підключається до сервера.

Робочий набір – це ваша локальна папка де ви можете вносити свої зміни.

Транк (Trunk) – первинне розташування коду в репозиторії.

Основні дії та атрибути систем контролю версій:

1. Додати – первинне додання нових даних до системи контролю версії
2. Ревізія – версія файлу в репозиторії.
3. Вершина (head) – остання ревізія файла в репозиторії.
4. Реєстрація - завантаження файла що було змінено до репозиторію.
5. Завантаження– отримання файла із репозиторіях.
6. Повідомлення при реєстрації – повинно описувати зміни що було внесено до файла.
7. Історія змін– це перелік усіх змін що було внесено до файла з моменту його додання в систему контроля версій.
8. Оновлення – це завантаження усіх файлів для отримання іх останньої версії.
9. Скасування - виконується коли потрібно відхилити локальні зміни та завантажити останню версію із репозиторію.

Додаткові дії:

1. Розгалуження – створення окремої копії файла чи папки для приватного використання (тестування, виправлення помилок, т.і.)
2. Пошук різниці між двома версіями файла.
3. Злиття – застосування змін з одного файла до другого.
4. Конфлікт це коли декілька змін до одного файла суперечать один одному.

5. Резолюція – виправлення змін що суперечать один одному та реєстрація файла в потрібну версію.

6. Блокування – використовується для контролю за змінами файла таким чином, що ніхто інший не зможе його змінити доки ви не знімете блокування.

7. Зняття блокування – це насильне розблокування файла для внесення змін до нього.

Team Foundation Server (TFS) містить потужну корпоративну систему контролю версій. TFS використовує базу даних Microsoft SQL Server у якості репозиторія, вона тразакційна та атомарна.

Головні особливості системи контролю версій TFS:

1. Робочі області.
   a. Області на вашому жорсткому диску де ви можете робити зміни.
2. Реєстрація\Завантаження.
   a. Завантаження позначає початок внесення змін.
   b. Реєстрація вносить ваші зміни обратно до репозіторія.
   c. TFS дозволяє спільне завантаження.
3. Набори зімн.
   a. Група змін при реєстрації файлів.
4. Ізольована реєстрація (shelving).
   a. Схожа на звичну реєстрацію.
   b. Зміни реєструються на сервері.
   c. Зміни не додаються до основного дерева коду проекту.
5. Розгалуження.
   a. Використовується для управління різними версіями продукту.

Методичні вказівки

1. Підключитись до Team Foundation Server.
2. Вивчити структуру системи контролю версій.
3. Створити робочю область та зв’язати локальну папку с серверною.
4. Звантажити останню версію проекту.
5. Реалізувати необхідний сценарій згідно варіанта, виданого викладачем.
6. Завантажити локальні зміни до репозиторію використовуючи реєстрацію та злиття.

Запитання для самоконтролю

1. Що таке система контролю версії та чим вона може бути корисною?
2. Які основні компоненти системи контролю версій?
3. Які основні операції можна виконувати в системі контролю версій?
4. Що таке робоча область в TFS?
5. Що таке ізольована реєстрація?
6. Що таке розгалуження та злиття? Навіщо використовувати розгалуження та злиття?

Посилання: [1], [5].

Завдання 2. Дослідження роботи з робочими елементами, сценаріями, задачами й помилками

Мета вивчити основи управління процесом розробки програмного забезпеченняз використанням Team Foundation Server та Microsoft Solution Framework (MSF) для Scrum. Розглянути повсякденну роботу розробника програмного забезпечення з використанням робочих

елементів, задач та сценаріїв. Вивчити різні підходи до оцінки обсягу робіт. Вивчити процеси управління помилками та їх виправлення.

Задачі

1. Вивчити використання сценаріїв для опису вимог в Team Foundation Server.
2. Вивчити декомпозицію сценаріїв у задачі.
3. Навчитись призначати завдання певному члену команди.
4. Вивчити життєвий цикл помилки.
5. Використовуючи видані викладачем вимоги, описати сценарій та декомпозувати його у задачі.

Теорія

Scrum є гнучким та легким процесом для керування процесом розробки програмного забезпечення.

Це оболонка для існуючих інженерних практик та є:

- Командо-орієнтованим підходом для ітеративної, поступової розробки програмних продуктів коли вимоги швидко змінюються;
- Це процес що контролює хаос конфліктних інтересів та потреб.

Scrum може бути розглянуто як паттерн.

Scrum найбільш адаптований для невеликих команд розробників (до 10 осіб), обмежений у часі та використовує спринти від одної до чотирьох неділь. Спринт це ітерація фіксованої довжини, результатом якої є видимий та корисний продукт (функціональність).

Правила спринта:

1. Тотальний фокус – ніяких небажаних відхилень від запланованої роботи протягом спринта.
2. Ніяких перерев чи зовнішніх змін.
3. Нові завдання що з’являються протягом спринта можуть бути невиконаними командою.

На початку спринта команда проводить нараду для планування робіт. На нараді команда обговорює вимоги (історії та сценарії), надає сценаріям пріорітет, оцінює їх та обирає ті що будуть реалізовані протягом спринта відповідно до ємності команди.

Протягом спринта команда працює так, щоб задовільнити наступним вимогам:

1. Проводити часті, короткі Scrum (stand up) наради.
2. Проводити видимі та придатні для користування інкременти.
3. Кожний наступний інкремент будується на базі попереднього.
4. Усі результати та обв’язки чітко визначаються.

Перед началом спринта:

1. Проводиться статусна нарада зі спонсорами та зацікавленими сторонами.
2. Інкременти поставляються замовнику.
3. Надаються повідомлення про усі проблеми та несподівані сітуації.
4. Наприкінці спринта усе може змінитися(роботу може бути додано, видалено, змінено пріоритети, проект може бути закритий).
5. Готуються нові оцінки та завдання для наступного спринта.

Вимоги

Хороший набір вимог має бути визначений з кількох точок зору. Існує декілька типів вимог: бізнес-вимоги, користувацькі вимоги, функціональні та вимоги по якості обслуговування.

Бізнес-вимоги визначають те що є важливим для успіху проекту. Як правили бізнес-вимоги представляють інтереси спонсорів проекту.

Вимоги користувачів описують завдання що вони повинні мати можливість зробити користуючись продуктом.

Функціональні вимоги або функціональні характеристики – це те що розробники повинні побудувати для того щоб задовільнити інші вимоги. Функціональні вимоги зазвичай визначаються дуже докладно, щоб допомогти розробникам зрозуміти що саме вони повинні розробляти.

Вимоги до якості обслуговування визначають договірні чи нефункціональні вимоги до системи. Зазвичай вони не представляють проблем конкретного користувача, скоріше це вимоги до продуктивності, масштабованості, безпеки системи та стандартів.

Сценарії використання та вимоги

Сценарії використання та вимоги це не одне й теж саме. Сценарії використання це UML (Unified Modeling Language) моделі, що описують набір кроків користувача для виконання конкретної задачі. Вимоги визначають що має бути розроблено для того щоб задовольнити користувача. Разом вони забезпечують якісну картину того як користувач бачить систему.

Для того щоб правильно та ефективно спланувати спринт всі сценарії використання повинні бути оцінені, щоб розподілити робоче навантаження між усіма членами команди. Оцінка визначає скільки приблизно зусиль треба витратити для того щоб задовільнити конкретним вимогам. Існує декілька заходів та методів оцінки. Послідовний підхід описує кількість рядків програмного коду, чи функцій. В гнучких методах розробки використовують бали сценаріїв (storypoint) та максимально продуктивні людино-дні (perfectmanday).

Оцінка в максимально продуктивних людино-днях визначає – як довго треба працювати якщо:

1. працівник зосереджений на одному завданні;
2. немає відволікаючих факторів;
3. працівник має усе що необхідно для роботи.

Бал сценарію це ‘розмір’ завдання, що формується під впливом того наскільки воно важке та трудоємне.

Відстеження та виправлення помилок це процес управління дефектами програмного забезпечення.

Дефекти можуть бути знайдені та зареєстровані в системі контроля любим із членів команди.

Дефект повинен бути якомога детально та повно описаний для того щоб команда могла зрозуміти його джерело, причини та вплив на систему.

Дефекти потім призначаються відповідному члену команди для виправлення. У рамках призначення розглядається також поточне навантаження на потенційного робітника. Коли призначення зроблено задача на виправлення дефекту додається в робочу чергу члена команди.

На наступному кроці дефект усувають (він може бути виправлений, відкладений або може бути обрано рішення не виправляти його взагалі) та закривають.

Розробка програмного забезпечення може бути більш легкою, команди можуть працювати більш ефективно якщо використовуються артефакти що запропоновано в шаблонах процесів Team Foundation Server. Team Foundation Server містить декілька шаблонів готових до використання в залежності від конфігурації процесів та потреб команди та проета. Ці шаблони: MSF for CMMI, MSF for Agile та MSF for Scrum.

Agile шаблони містять наступні артефакти: робочі елементи, звіти, робочі книги та приладні дошки. Команди можуть використовувати робочі елементи для відстеження робочої інформації, аналізу прогресу, та приймати відповідні рішення.

Шаблон Agile визначає наступні типи робочих елементів:

1. Користувацькі сценарії
2. Задачі
3. Сценарії тестування
4. Спільні етапи
5. Дефекти
6. Проблеми.

Користувацькі сценарії. Команда створює користувацькі сценарії для опису можливостей, функцій та вимог шо треба реалізувати. Користувацький сценарій описує цілі замовника на загальному рівні та є фундаментальним елементом етапа планування, тому що він допомагає команді оцінювати, визначати пріоритет, визначати, планувати та веріфікувати роботу що відноситься до кожного користувацького сценарію.

Задачі. Команда визначає задачі для відстеження роботи що потрібно виконати для реалізації користувацького сценарію, або інших напрямків роботи що визначені для проекту. Задача повинна представляти невеликий об'єм роботи що може бути виконано протягом одного-двох днів. Більш великі задачі можуть бути розділено на підзадачі. Шляхом відстеження робочого часу для кожної із задач, команда отримує чітку картину стану роботи та прогресу розробки.

Сценарії тестування. Використовуються командою для визначення тестів що потрібно виконати для підтримки користувацьких сценаріїв. Можна визначити тести що виконуються вручну, вказавши послідовність дій та перевірок, або реалізувати автоматичні тести.

Спільні етапи. Необхідні команді для впорядкування тестових сценаріїв та підвищення спроможності їх обслуговування. В спільних етапах можна визначити послідовність дій та перевірок для повторного використання в тестових сценаріях. Більшість тестів можуть містити туж саму послідовність дій, користуючись спільними етапами команда взмозі використовувати її у множині тестових сценаріїв (наприклад аутентифікація користувача в системі може бути реалізована як спільній етап).

Дефекти. Команда відстежує помилки в програмному коді та може реєструвати їх як робочі елементи типу дефект. Реєструючи дефект ви взмозі точно звітувати о подробицях помилки та допомогти команді зрозуміти її причини та вплив на систему. При реєстрації дефекта необхідно описати перелік дій що привели до небажаної поведінки системи, це допоможе іншим відтворити дефект пізніше. Результати тестування повинні чітко вказувати на проблему. Чіткість а повнота опису часто впливає на ймовірність того, що помилку буде виправлено.

Проблеми. Ви можете визначити відомі або потенційні проблеми, перешкоди чи ризики для вашого проекту реєструючи робочий елемент «Проблема» в Team Foundation Server. Якщо необхідні конкретні дії, проблема може транслюватись у одну чи декілька задач що потрібно виконати для їх усунення. Наприклад, технічна проблема чи невизначеність може транслюватись в задачі по створенню архітектурного прототипу. Команди повинні завжди закликати своїх членів до пошука потенційних проблем та переконатися що вони надають якомога більше інформації о проблемах що можуть затримати прогрес команди.

Методичні вказівки

1. Вивчить процесну документацію на командному порталі TFS.
2. Розглянути шаблон документа для опису користувацького сценарію.
3. Користуючись вимогами, що видані викладачем опишіть користувацький сценарій.
4. Оцінить трудоємність реалізації сценарію.
5. Розкладіть сценарій на задачі.

Вопроси для самокнтролю

1. Що таке Scrum?

2. Що таке спринт?
3. Що таке нарада з планування спринту?
4. Які типи вимог вам відомі?
5. В чому різниця між користувацьким сценарієм та вимогами?
6. Що таке бали сценарію та максимально продуктивні людино-дні?
7. Опишіть життєвий цикл дефекта програмного забезпечення.

Посилання: [1]; [2]; [3]; [4].

Завдання 3. Робота із запитами до робочих елементів, створення збірки

Мета дізнатися як працювати з запитами до бази робочих елементів, як створювати власні запити. Вивчити основи автоматизованих збірок та навчитись створювати та кофігурувати збіркі.

Задачі

1. Вивчити концепцію запитів до бази робочих елементів в Team Foundation Server.
2. Дізнатись як користуватися, створювати, та конфігурувати запити.
3. Вивчити основи автоматизації збірок.
4. Навчитися створювати та конфігурувати процеси автоматизованої збірки в Team Foundation Server.

Теорія

Робочий елемент – це запис у базі даних що створюються в Team Foundation Server для фіксації опису задачі, відповідальної особи, пріорітету та стану прогреса. За рахунок визначення індивідуальних робочих елементів та їх зберігання у загальній базі даних та складі метрік, команда має можливість відповісти на питання про стан проекту по мірі їх виникнення. Робочі елементи, зв'язки між ними та файли додатків зберігаються в базі даних для відстеження їх стану та змін що вносяться. (Рис 1).

Рис1

Для відстеження статусу робочого елементу, прогресу ітерації чи реліза продукта, ви можете визначити та виконувати запити до робочих елементів. В Visual Studio Team Explorer ви можете знайти дві різні папки для зберігання запитів до бази даних робочих елементів: Командні

Запити та Мої Запити. Папка Командні Запити містить набір заздалегідь визначених запитів що є спільними для всієї команди розробки. Створіть новий запит в папці Командні Запити якщо ви бажаєте щоб він був доступний для всієї команди. Папка Мої Запити призначена для зберігання ваших персональних запитів та не містить нічого за замовченням. Для того щоб створити спільний чи персональний запит ви можете використовувати наступний робочий процес(Рис 2).

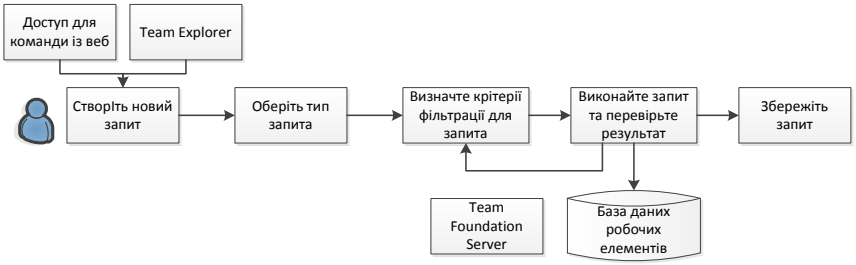

Рис2

Використовуйте запити для пошуку робочого елемента на основі інформації що зберігається в базі даних. Простий запит може шукати робочі елементи що містять певне значення в одному полі форми, наприклад усі робочі елементи що назначені вам на виконання. Більш складні запити можуть виконувати пошук по декільком полям форми робочого елемента. Для створення запиту необхідно користуватись умовами для поліпшення пошукових результатів.

Оператор запиту використовується в умові для створення запиту до бази робочих елементів. Кожна умова запиту складається з імені поля, оператору запита та значення.

Таблиця 1 містить перелік операторів запиту що доступні в системі відстеження робочих елементів Team Foundation Server.

Таблиця1

Оператор запиту	Опис	Відповідні типи полей
=	Вертає робочий елемент якщо значення поля відповідає заданому.	Число, Текст, Дата, Дерево
<>	Вертає робочий елемент якщо значення поля не відповідає заданому.	Число, Текст, Дата, Дерево
>	Вертає робочий елемент якщо значення поля більше ніж задане.	Число, Текст, Дата, Дерево
<	Вертає робочий елемент якщо значення поля менше ніж задане.	Число, Текст, Дата, Дерево
>=	Вертає робочий елемент якщо значення поля більше чи дорівнює заданому.	Число, Текст, Дата, Дерево
<=	Вертає робочий елемент якщо значення поля менше чи дорівнює заданому	Число, Текст, Дата, Дерево
Contains	Вертає робочий елемент якщо значення поля містить заданий текст	Текст
Does Not Contain	Вертає робочий елемент якщо значення поля не містить заданий текст	Текст
In	Вертає робочий елемент якщо значення поля містить будь-яке значення із заданого набору. Наприклад, порівняння поля ID з набором значень 100, 101, 102 поверне робочі елементи 100, 101, та 102.	Число, Текст, Дата
Was Ever	Виконує пошук по історії поля. Вертає робочий елемент якщо будь-яке історичне значення поля співпадає із заданим значенням.	Текст, Дата

Оператор запиту	Опис	Відповідні типи полей
Under	Виконує пошук по ієрархії та вертає робочі елементи що є наслідниками вузла зазначеного параметром.	Дерево
Not Under	Виконує пошук по ієрархії та вертає робочі елементи що не є наслідниками вузла зазначеного параметром.	Дерево

Змінні дозволяють генерувати параметри, що дає змогу створювати динамічні запити, що виконують пошук по поточній даті, користувачу чи проекту.

Таблиця 2 містить перелік змінних запиту що досупні для використання в системі відстеження робочих елементів Team Foundation Server.

Таблиця2

Змінна запиту	Результат
@Me	Використовуйте змінну @Me для автоматичного пошуку для поточного користувача, якшо поле містить ім'я користувача. Наприклад, якщо ви хочете знайти усі робочі елементи що вами відкриті потрібно встановити поле Activated By, з оператором =, та значення @Me.
@Project	Використовуйте змінну @Project для будь-якого поля що має посилання на проект для пошуку по поточному проекту. Наприклад, якшо ви хочете знайти перелік усіх активних робочих елементів для обраного проекту необхідно обрати поле Team Project, оператор =, та значення @Project.
@Today	Використовуйте змінну @Today для будь якого поля типу Дата, для пошуку по поточній даті. Запит буде використовувати поточну дату в момент його виконання. Ви також можете модифікувати змінну @Today додаючи чи віднімаючи дні. Наприклад, для пошуку усіх робочих елементів що було активовано за останні 7 діб, потрібно встановити поле Activated Date, оператор>=, та значення @Today - 7.

Збірка проектів з використанням Team Foundation Server

Організації та команди потребують повторюваний та надійний метод для створення збірок, доступних громадськості на регулярній основі. Процеси життєвого циклу розробки програмного забезпечення еволюціонують с пливом часу. Команди розробки зрозуміли важливість наявності періодичних збірок продуктів. Спочатку усе починалося з недільних збірок. Потім графік став більш щільним коли було запропоновано практику нічних збірок та навіть більш щільним з введенням погодинних графіків збірки продуктів. Коли переваги частих збірок стали очевидними для організацій, вони захотіли робити це якомога частіше. Отже зараз ми маємо безперервну інтеграцію. Але нічні збірки все ще дуже популярні тому що команди покладаються на них для отримання формальних, надійних збірок.

Безперервна інтеграція – це процес генерації збірки в той самий момент як програміст реєструє зміни коду в системі контроля версій.

Team Foundation Build пропонує функціональність для створення лабораторії збірки, та є счастиною Visual Studio Team Foundation Server. Користуючись Team Foundation Build, ви маєте можливість синхронізувати похідний код, компілювати застосування, виконувати необхідні юніт тести, аналіз коду та публікувати звіти про всі ці дії. Дані про збірку зберігаються в базі даних для історії та звітів. Team Foundation Build працює з іншими інструментами Team System

протягом процесу збірки, в тому числі з системою контролю версій, базою даних робочих елементів, та інструментами тестування.

Team Foundation Build розроблено для використання с Team Foundation в розподіленому середовищі як показано на рис 3.

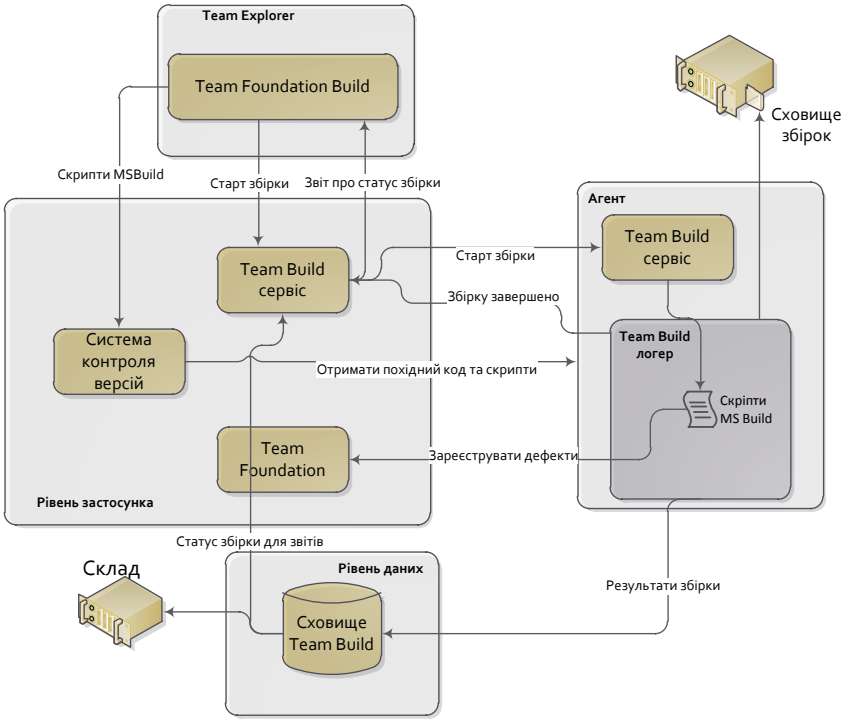

Рис3

Team Foundation Build взаємодіє с Team Explorer. Team Foundation Build розглядає визначення збірок як частину командного проекту. Визначення збірок можна побачити користуючись Team Explorer в папці Team Builds. Такі операції як запуск збірки, створення нової збірки можна виконати користуючись панелью Team Explorer в Visual Studio. Різні визначення збірок можуть існувати в папці Team Builds для кожного командного проекту.

Для створення простішого визначення збірки користуйтесь наведеними нижче кроками:

1. В Visual Studio Team Explorer оберіть папку Build, клацніть правою кнопкою миші та оберіть пункт меню New Build Definition.
2. Введіть ім’я нового визначення збірки.
3. Оберіть робочу область.
4. Оберіть буде виконувати збірку та введіть адресу сховища куди буде покладений результат роботи.
5. Оберіть необхідні опції що будуть ініціювати збірку.
6. Клацніть кнопку OK для збереження нового визначення збірки.

Методичні вказівки

1. Розгляніть існуючі в Visual Studio Team Explorer запити до бази даних робочих елементів.
2. Користуючись варіантом, виданим викладачем, створіть та виконайте приватні та командні запити.
3. Користуючись варіантом, виданим викладачем, створіть нове визначення збірки вказавши вашу робочу станцію в якості агента.
4. Ініціалізуйте процес збірки та розгляніть згенеровані звіти.

Питання для самоконтроля

1. Що таке робочий елемент та як він зберігається в Team Foundation Server?
2. Для чого команди користуються базою даних робочих елементів?
3. Які типи робочих елементів наявні в шаблоні MSF для Agile?
4. Поясніть покликання різних типів робочих елементів в шаблоні MSF для Agile.
5. Що таке запит робочих елементів? Які типи запитів ви знаєте?
6. Що таке безперервна інтеграція та для чого її використовують?

Посилання: [1].

Завдання 4. Аналіз стану проекту за допомогою звітів, аналіз коду, профілювання застосування

Мета вивчити інструменти відстеження життєвого циклу розробки програмного забезпечення наявні в Microsoft Team Foundation Server; вивчити інструменти та методи аналізу коду; навчитись профілювати застосування та оптимізувати його продуктивність та використання пам'яті.

Задачі

1. Вивчити інструменти моніторингу проекта.
2. Навчитись аналізувати дані наявні в звітах Team Foundation Server.
3. Навчитись аналізувати похідний код користуючись Microsoft Visual Studio.
4. Навчитись конфігурувати та виконувати сесії профілювання, аналізувати проблеми та виправляти її для поліпшення продуктивності застосунка та знизити використання їм пам'яті.

Теорія

Ви можете аналізувати прогрес та якість вашого проекту користуючись звітами що їх пропонують сервіси SQL Server Reporting. Ці звіти поставляються із шаблонами процесів наявними в Team Foundation Server (наприклад MSF for Agile Software Development). Ці звіти агрегують метрики з бази даних робочих елементів, системи контроля версій, результатами тестування, та збірок та допомагають відповісти на питання про стан речей на проекті.

Більшість цих звітів пропонують різноманітні фільтри шо можна використовувати для параметризації контенту що буде включено до звіту. Фільтри містять періоди часу, ітерації та області проекта, типи робочих елементів та їх стан. Питання на які вони відповідають відносяться до усіх типів робочих елементів як то: користувацькі сценарії, сценарії тестування, задачі та дефекти.

Усі наявні звіти можна поділити на декілька категорій:

1. Відстеження дефектів.
2. Відстеження збірок.
3. Відстеження стану проекта.
4. Виявлення доданої роботи.
5. Відстеження процесів тестування продукта.

Відстеження дефектів

Звіт статусів повідомлення про дефект може бути використаний для відповіді на наступні питання:

- Чи достатньо швидко команда виправляє дефекти шоб встигнути вчасно?

- Чи команда виправляє більш пріоритетні дефекти в першу чергу?
- Яке розподілення дефектів по пріоритету та важкості виправлення?
- Скільки дефектів призначено до виправлення кожному члену команди?

Звіт про тенденції дефектів відстежує з якою швидкістю команда виявляє та виправляє дефекти програмного забезпечення. Цей звіт демонструє зміну середньої кількості дефектів, що зареєстровано, виправлено та закрито с пливом часу. Ви можете використовувати цей звіт для відповіді на наступні питання:

- Яку кількість дефектів команда реєструє, виправляє, та закриває протягом дня?
- Який загальний тренд обробки дефектів для команди?
- Чи знижається кількість реєстрації та виправлення дефектів наприкінці ітерації, як це повинно бути?

Кількість дефектів може бути різною в залежності від того в якій стадії життєвого циклу розробки програмного забезпечення знаходиться проект. Команда повинна знаходити менше дефектів протягом перших ітерацій ніж в останніх. Команда повинна закривати більшість дефектів в ітераціях що знаходяться ближче до кінця життєвого циклу продукта. Коли продукт стабілізується наприкінці життєвого циклу команда повинна знаходити меншу кількість дефектів.

Звіт реактивації дефектів демонструє наскільки ефективно команда виправляє дефекти та може бути використаний для відповіді на наступні питання:

- Яка кількість дефектів була реативована?
- Яку кількість користувацькі сценаріїв було реактивовано?
- Чи команда знаходить та виправляє реактивовані дефекти з достатньою швидкістю?

Загальний прийнятний рівень реактивації робочих елементів повинен бути на рівні 5% або менше та не повинен збільшуватися протягом ітерації.

Відстеження збірок

Індикатори якості збірки демонструють покриття кода тестами, міграцію коду, та кількість дефектів для заданого визначення збірки. Цей звіт можна використовувати для того щоб зрозуміти як близько до фінальної якості знаходяться ті чи інші порції коду. Його використовують для відповіді на наступні запитання:

- Яка якість програмного продукту?
- Як часто виконуються тести та яку частину коду було протестовано?
- На основі метрик коду та тестів, чи взмозі команда задовільнити цілі проекту?

Підсумковий звіт збірки пропонує інформацію о результатах тестування, покриття коду тестами, міграції коду, та якості кожної збірки. Міграція коду обчислюється за рахунок визначення кількості рядків коду що було додано, видалено або змінено командою, поділена на загальну кількість рядків коду в збірці.

Успіх збірки с пливом часу демонструє ілюстровану версію підсумкового звіту. Він демонструє статус останньої збірки для кожної категорії та для кожного дня. Цей звіт можна використовувати для відстеження якості коду що команда реєструє в системі контроля версій.

Відстеження стану проекта

Burn down та burn rate звіти показують тенденцію виконаної роботи та роботи що залишається протягом заданого відрізку часу. Burn rate забезпечує розрахунки швидкості з якою була виконана робота та необхідної швидкості для виконання роботи що залишилась для заданого відрізку часу. Крім того, діаграма показує кількість робочих часів призначених членам команди що виконано та залишилось. Він допомагає зрозуміти:

- Чи встигне команда закінчити ітерацію вчасно?
- Чи виконає команда роботу на основі поточної швидкості розробки?
- Яку кількість роботи має кожний член команди?

Звіт про роботу що залишилась використовується для відстеження прогресу після того як команда оцінила свої задачі та робота почалась. Він допомагає зрозуміти:

- What is the cumulative workflow?
- Чи встигне команда закінчити ітерацію вчасно?
- Чи зростає кількість робочих задач протягом ітерації?
- Чи команда має занадто багато роботи в прогресі?
- Наскільки точно команда дає оцінки своїм задачам?

Позитивний звіт демонструє стабільний прогрес у виконанні та закритті робочих завдань, прямокутна форма звіту вказує, що очікувана кількість витрат робочого часу близька до необхідного обсягу робіт.

Оглядовий звіт користувацьких сценаріїв містить перелік усіх сценаріїв, от фільтрований по ітераціям та робочим областям та допомагає знайти відповіді на такі питання:

- Яка необхідна кількість витрат робочого часу для кожного користувацького сценарію?
- Яку кількість роботи вже було виконано?
- Чи були виконані тести для сценарію?
- Яку кількість активних дефектів має кожен сценарій?

Статус усіх ітерацій показує командний прогрес напротязі декількох ітерацій. Переглядаючи цей звіт можна відстежувати продуктивність команди протягом декількох успішних ітерацій. Він надає відповіді на такі питання:

- Наскільки стійкий прогрес має команда протягом декількох ітерацій?
- Яку кількість користувацьких сценаріїв команда виконала протягом кожної ітерації?
- Яку кількість дефектів команда виявила, виправила та закрила протягом кожної ітерації?

Позитивний звіт демонструє зростання продуктивності команди, в той же час оцінки необхідних витрат робочого часу повинні становитись більш точними та наближатись до реально витрачених часів.

Виявлення доданої роботи

В основному, згідно правил, протягом ітерації що була спланована та знаходиться у процесі виконання не повинно виникати незапланованої додаткової роботи. Наявність незапланованої роботи може бути прийнятна, особливо якщо команда запланувала деякий буфер що є достатнім для обробки додаткового навантаження (наприклад виправлення дефектів). З іншого боку незапланована робота може представляти реальну проблему якщо команда не має потенціалу для її виконання та змушена урізати заплановану роботу. Звіт про незаплановану роботу інформує про те яку кількість роботи що не було заплановано додано протягом ітерації. Він вимірюється як кількість доданих робочих елементів.

Моніторинг процесу тестування

Дуже важливо чітко розуміти коли роботу «виконано». Термін «виконано» визначається критеріями прийнятності що має кожен користувацький сценарій. Також повинна бути визначена та реалізована стратегія тестування для перевірки чи відповідає похідний код критеріям

прийнятності. Звіти тестування можуть бути корисними для відстеження того як команда реалізує тестові сценарії та як вони покривають користувацькі сценарії протягом розробки.

Звіт про наявність тестів демонструє:

- Коли усі тести будуть готові для виконання та чи будуть вони готові наприкінці ітерації;
- Яку кількість тестів команда повинна розробити та переглянути;
- Яка кількість тестів готова для виконання.

Позитивний звіт демонструє стабільне зростання розроблених тестів.

Звіт по плану тестування. Після того як команда розробила тести та почала їх виконувати, цей звіт використовується для відстеження прогресу тестування продукта. Користуючись цім звітом ви можете зрозуміти:

- Яка частка тестування завершена;
- Чи встигне команда закінчити тестування вчасно;
- Яку кількість тестів лишилось виконати;
- Скільки тестів пройдено успішно, не успішно та скільки тестів заблоковано?

В ідеалі, звіт демонструє відносно стабільну кількість тестів одночасно із стабільним ростом тестів що виконуються та що є успішними. В той час як цикл розробки прогресує кількість тестів в статусі «Пройдено» повинна зростати, в той час як кількість тестів в інших статусах повинна зменшуватись.

Огляд аналізу коду

В версіях Visual Studio Premium та Visual Studio Ultimateви можете використовувати вбудовані інструменти аналізу для дослідження якості похідного коду та виявляти потенційні проблеми, такі як незахищений доступ к даним, порушення використання та проблеми дизайну програмного продукту. Інструменти аналізу коду працюють із платформою Microsoft .NET Framework, натівними (C чи C++) застосунками та базами даних.

Інструменти аналізу керованого коду досліджують керовані збірки та надають інформацію про ці збірки, таку як порушення правил програмування та дизайну що визначені у загальних принципах дизайну застосувань Microsoft .NET Framework.

Як розробник ви можете виконувати аналіз коду автоматично кожен час коли ви компілюєте проект, чи вручну.В той же час аналіз коду може бути включений в правилах реєстрації коду Team Foundation Server, таким чином він буде виконуватись перед кожною спробою зареєструвати зміни в системі контроля версій.

Усі правила згідно яких виконується аналіз коду згруповані у набори. Ви може користуватись одним із стандартних наборів Microsoft чи сформувати власний набор правил вашого проекту що задовольняє необхідним вимогам.

Інструменти аналізу коду представляють перевірки що виконуються та представлені як попередження. Кожне попередження визначає відповідну проблему в дизайні чи реалізації та, коли це можливо, подає інформацію про те як виправити зазначену проблему.

Часто може бути корисно вказати що попередження не повинно бути застосованим. Таким чином ви як розробник можете проінформувати других членів команди що можуть переглядати ваш код, що попередження було проаналізовано та проігноровано чи пригнічено. Пригнічення попереджень реалізовано у вигляді користувацьких атрибутів. Для пригнічення попередження додайте атрибут *SuppressMessage*до похіного коду як показано далі.

```
[System.Diagnosis.CodeAnalysis.SupressMessage(“Microsoft.Design”,
“CA1039:ListsAreStrongTyped”)]
Public class SomeClass
{
        // implementation
}
```

Атрибут *SuppressMessage* може бути використаний на різних рівнях: збірка, модуль, тип, член, чи рівні параметрів.

Профілювання застосунків.

Профайлер – це інструмент що відстежує процес виконання іншого застосунка. Профайлер загального середовища віконня (Common Language Runtime) це динамічна бібліотека класів (DLL), яка складається з функцій що отримують та відправляють повідомлення до CLR за допомогою програмного інтерфейсу профілювання. Збірка профайлера завантажується CLR під час виконання.

Традиційні інструменти профілювання фокусуються на вимірюванні виконання застосунка. Тобто, вони вимірюють час що затрачений у кожній функції чи використання застосунком пам'яті з пливом часу. Програмний інтерфейс профілювання націлений на біль ширший клас інструментів діагностування, наприклад, утиліти для вимірювання покриття коду тестами. Програмний інтерфейс профілювання не тільки вимірює але й відстежує виконання застосунка. З цієї причини, профілювання ніколи не повинно використовуватись самим застосунком, та процес виконання застосунка не повинен бути залежним від профайлера, чи знаходитись під його впливом.

Профілювання CLR застосунка потребує більше підтримки ніж профілювання традиційного застосунка що скомпілювано у машинний код. Це тому, що CLR вводить такі концепції як домени застосунків, збірка сміття, керована обробка виняткових ситуацій, компіляція коду на льоту(just-in-time compilation - перетворення проміжної мови Microsoft чи (MSIL) у машинно залежний код), та іншні. Звичайний механізм профілювання не взмозі ідентифікувати чи надати корисну інформацію про ці можливості. Програмний інтерфейс профілювання надає цю відстню інформацію дуже ефективно, з мінімальним вливом на продуктивність загального середовища виконання та застосунка що профілюється.

Компіляція коду на льоту надає гарні можливості для профілювання та дозволяє модифікувати проміжний код у пам'яті в цілях його вимірювання та відстеження перш ніж він буде скомпільований.

Інструменти профілювання Microsoft Visual Studio надають п'ять методів що можуть бути корисні для сбору даних о продуктивності застосунка:

1. Вибірки – метод збирає дані про роботу що виконана застосунком.
2. Контрольне-вимірювання – збирає детальну інформацію про те скільки часу було витрачено протягом виклику кожної функції.
3. Паралелізм збирає докладну інформацію про багатопотокові додаткі.
4. Вимірювання пам'яті - збирає інформацію про використання застосунком пам'яті та збирання зміття.
5. Взаємодія рівнів – збирає інформацію про синхронний виклик функцій ADO.NET до бази даних SQL Server.

Таким чином інструменти профілювання Visual Studio Profiling допомагають ідентифікувати проблеми із продуктивністю в похідному коді та порівнювати продуктивність декількох можливих рішень. Помічник інструментів профілювання з параметрами по замовчанню може швидко надати розуміння щодо більшості проблем з продуктивністю застосунка. Можливості та налаштування інструментів профілювання надають щільний контроль над процесом. Цей контроль включає точне фокусування на необхідних частинах коду, збір даних про час виконання на рівні блоків та включення додаткових даних із метриками процесора та системи.

Наступні кроки складають основний процес використання інструментів профілювання:

1. Створіть новий сеанс вимірювання продуктивності вказав метод збору інформації та дані що вам потрібні.
2. Зберіть необхідну інформацію шляхом виконання застосунка в сеансі профілювання.
3. Проаналізуйте зібрані дані на предмет проблем продуктивності.

4. Модифікуйте код для підвищення продуктивності застосунка.
5. Зберіть дані профілювання для модифікованого застосунка та порівняйте з попередніми результатами.
6. Згенеруйте звіт що задокументує підвищення продуктивності програмного продукта.

Методичні вказівки

1. Розгляніть звітні можливості Visual Studio Team Foundation Server.
2. Користуючись варіантом завдання, що видав викладач, проаналізуйте звіти та знайдіть потенційні проблеми.
3. Користуючись варіантом завдання, що видав викладач, розгляніть похідний код, виконайте аналіз коду інструментами Visual Studio, та зробіть необхідні виправлення.
4. Користуючись варіантом завдання, що видав викладач, розгляніть похідний код, пошукайте можливі проблеми с продуктивністю, налаштуйте та виконайте сессію профілювання.
5. Виправте проблеми якщо знайдете та виконайте профілювання ще раз, порівняйте результати.
6. Згенеруйте звіт по аналізу продуктивності застосунку.

Питання для самокнтролю

1. Які типи звітів в Team Foundation Server вам відомі?
2. Яка мета процесу моніторингу дефектів?
3. Яка мета процесу моніторингу компіляції?
4. Яка мета процесу моніторингу стану проекта?
5. Яка мета моніторингу доданої роботи?
6. Яка мета моніторингу процеса тестування?
7. Що таке аналіз коду та як він працює?
8. Що таке профілювання?
9. Які методи профілювання наявні в Microsoft Visual Studio?

Посилання: [1].

ПОСИЛАННЯ

1. Meier J.D., Taylor J., Mackman A., Bansode P., Jones K. Team Development With Visual Studio Team Foundation Server. – Microsoft Corporation, 2007. – 496 p.
2. Larman C. Agile & Iterative Development. – Addison Wesley, 2003. – 368 p.
3. Highsmith J., Agile Systems Development Ecosystems. - Addison Wesley, 2002. – 448 p.
4. Michael S. V. Turner Microsoft Solutions Framework Essentials: Building Successful Technology Solutions. – Microsoft Press, 2006. – 336 p.
5. Sidorov N. Introduction to Software Engineering. – NAU.- 2010.- 60 p.