KE Preparation Document

# Process, Terminology and Methodologies

## Testing Essentials

### **What is testing**

Це процес статичного та динамічного пошуку дефектів в роботі програми, який включає такі етапи як планування, підготовка, перевірка роботи продукту на те чи відповідає вона визначеним вимогам.

Error – поведінка програми не відповідає очікуванням коли її використовують неправильно.

Defect – поведінка програми не відповідає вимогам чи допущено помилку на стадії проектування чи складанні вимог. Може призвести до системних failure.

Failure – робота коду програми призводить до неправильної поведінки\краху системи загалом.

### **Why testing is necessary**

Тестування є необхідним оскільки ми всі робимо помилки і потрібно перевіряти створені нами продукти, щоб їх використання не привело до якихось серйозних негативних наслідків, починаючи від втрати грошей, закінчуючи смертю людей.

### **Main testing principles**

**Принцип 1: Тестування показує наявність помилок, проте не показує, що їх нема**

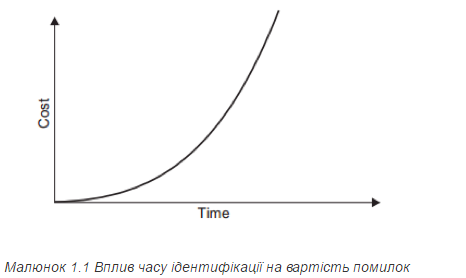
Запуск тесту за допомогою програмної системи може тільки показати, що існує один або кілька дефектів. Тестування не може показати, що програмне забезпечення без помилок. Незважаючи на те, що можуть бути інші цілі, як правило, основною метою тестування є знаходження дефектів. Тому тести повинні бути розроблені, щоб знайти якомога більше дефектів, як це можливо.

**Принцип 2: Всебічні випробування неможливо**

Неможливо сказати що ти протестував 100% кейсів. Проект є обмежений часом і бюджетом. Для того визначаються пріоритети і ризики для вибору тестових сценаріїв.

**Принцип 3: Попереднє тестування**

Тестування повинно починатись на початкових стадіях розробки і повинно бути сфокусованим на попередньо визначених цілях.



**Принцип 4: Дефект кластеризация**

Це застосування принципу Парето до тестування програмного забезпечення: приблизно 80 відсотків проблем знаходяться в приблизно 20 відсотків модулів. Це корисно, якщо тестування активності відображає це поширення дефектів, і призначається для областей застосування в рамках тесту, де велика частина дефектів можуть бути знайдені. Проте, слід пам'ятати, що тестування не повинно концентруватися виключно на цих частинах. Там може бути менше дефектів в решти коду, але випробувачі все ще потрібно шукати усередині для них.

**Принцип 5: Парадокс пестициду**

якщо постійно виконувати одні і ті ж тести регулярно, то нові баги не знаходитимуться. Потрібно періодично переглядати кейси, додавати нові. Шукати нові шляхи проходження виконуваних кейсів.

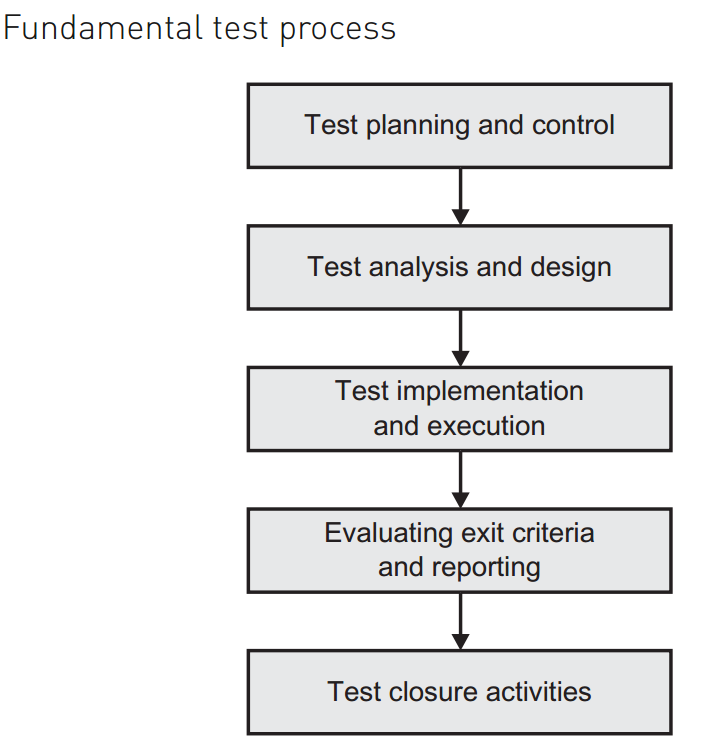
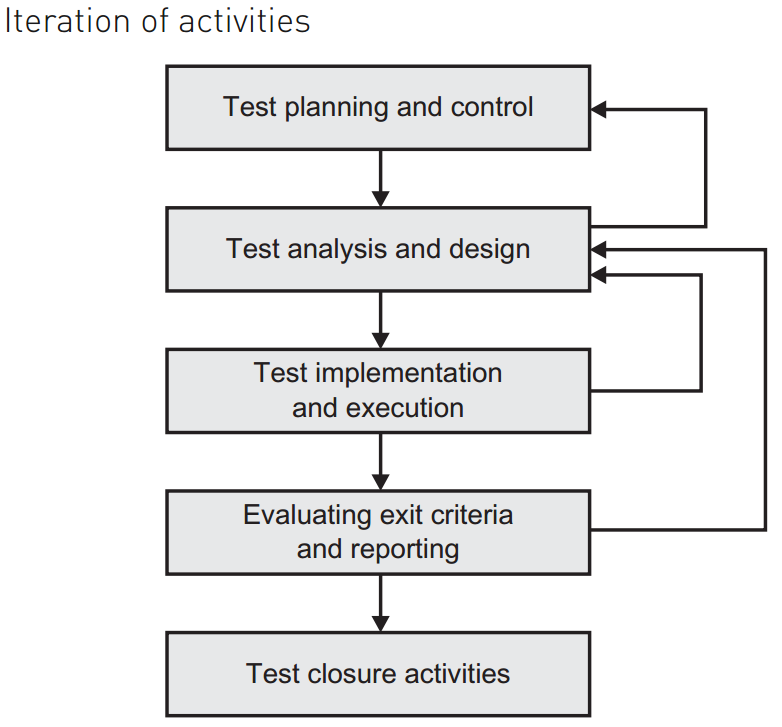
**Принцип 6: Тестування залежить від контексту**

Тобто тестування якихось медичних приладів відрізняється від тестування онлайн магазину

**Принцип 7: відсутність помилок це помилковість**

Програмне забезпечення без будь-яких відомих помилок не обов'язково готовий до відправки.  Той факт, що немає дефектів видають це не причина, щоб відправити програмне забезпечення.Пошук і фіксання дефектів не допоможуть, якщо програма не виконує потрібних користувачеві функцій.

### Key phases of Fundamental Test process according to ISTQB standard

2. Planning and control:
   * створення тестової стратегії і політики, врахування ризиків, визначення цілей тестування;
   * визначаються техніки тестування, рівень покриття, визначення очікуваних результатів для замовника, кількість людей та середовищ для тестування;
   * визначення definition of done;
   * потрібно впродовж всього тестування порівнювати план з поточним станом справ і інформувати керівників у випадку відхилень;
   * Оцінка результатів тестування та складання репортів.

* **Monitoring and Control:**
* **Відсоток роботи, виконаної в рамках підготовки тестового прикладу (або відсоток запланованих тестові приклади підготовлені)**
* **Відсоток роботи, виконаної в рамках підготовки тестового середовища.**
* **Виконання тесту випадок (наприклад, кількість тестових випадків працюють / не працюють, і тестові приклади пройшов / не вдалося).**

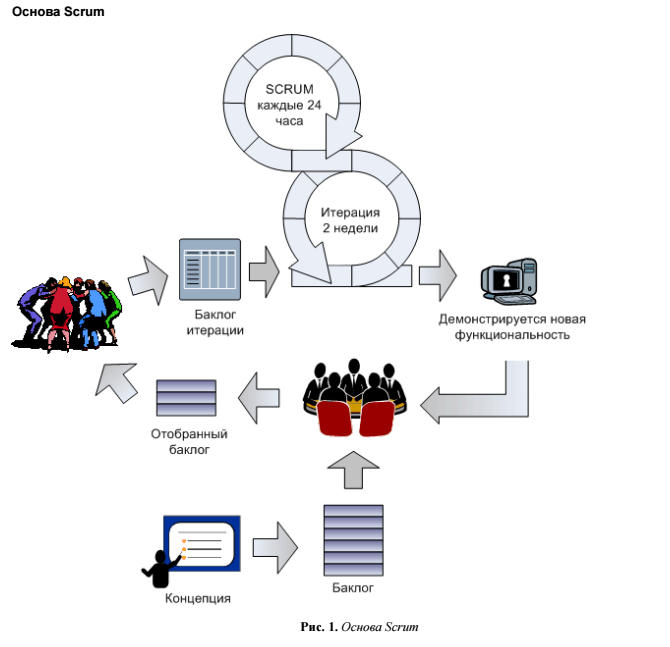
1. Analysis and design:
   * Створоюються матреальні артефакти тестування: тест дизайни
   * Рев\*ювається специфікація і технічна документація
   * Огляд вимог, архітектура, дизайн, специфікації інтерфейсів, які в сукупності складають основу тестування
   * Аналіз тестових завдань, специфікації
   * Проектування тестів, в тому числі призначення пріоритету
   * Визначити чи вимоги і система можна тестувати
   * Чи є будь-яка інфраструктура і інструменти для тестування
   * Створення двонаправленого простежуваності між тестової бази і тестових випадків.
2. Implementation and execution
   * Написання тест кейсів, їх пріоретизація, створення тестових даних
   * Налаштовуються тестові середовища
   * Збір тестових випадків в тестові набори, де тести можуть бути запущені один за іншим для підвищення ефективност
   * Виконання тест кейсів
   * Ведення тестового журналу виконання кейсів
   * Порівняння фактичних результатів з очікуваними результатами
3. Evaluating exit criteria and reporting
   * Перевірка тестових логів
   * Логування дефектів
   * Написання підсумкових репортів
4. Test closure activities
   * Перевірка чи всі дефекти були залоговані
   * Чи всі очікувані результати були виявленні
   * Фіналізація тестових артефактів для майбутнього використання
   * Аналіз процесів для покращення їх в майбутньому

### Fundamentals of SCRUM process

Scrum - одна з найпопулярніших методологій гнучкої розробки. Одна з причин її популярності - простота. Scrum по-справжньому простий, його можна описати в одній короткій статті.

Scrum це Agile фреймвок.

Agile - ітеративний підхід до ведення проектів, обмежений часовими рамками, який визначає кінцевий продук частинами, а не вцілому коли завршилась розробка. Він включає в себе кілька фреймворків для ведення проектів: DSDM, Scrum, XP, Kanban, Lean...

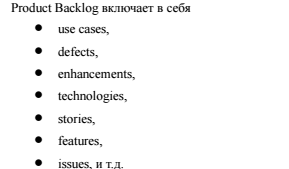


**SCRUM Roles**:

* **Product owner** - людина, яка представляє інтереси замовника. Він відповідає на запитання команди щодо вимог, визначає пріорітети юзер сторь.
* **SRUM master** - вирішення організаційних питань для дев-тіму. Він є драйвером на дейлі мітінгах, слідкує за порядком на дашборді і графіками.
* **Team** - люди, які безпосередньо роблять продукт.
* Де коли ще виділяють Stakeholders - люди, які зацікавлені проектом і мають до нього відношення (кінцеві користувачі, інвестори, менеджмент, супорт).

**Artifacts**:

* Product increment - шипаблонна версія продукту;
* Product Backlog - це пріоритезувати список наявних на даний момент бізнес-вимог і технічних вимог до системи.



* Sprint Backlog містить функціональність, обрану Product Owner з Product Backlog. Всі функції розбиті за завданнями, кожна з яких оцінюється командою.

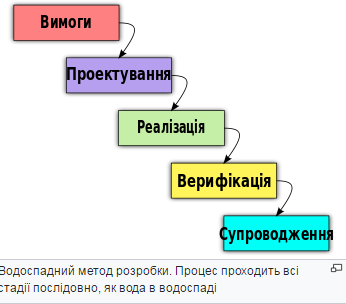
**Activities**:

* Release planning (backlog refinement);
* Sprint planning;
* Daily Scrum;
* Sprint review (demo of product);
* Sprint retrospective;

**Спринт (Sprint)**

У Scrum ітерація називається Sprint. Її тривалість становить 2 – 4 тижні. Результатом Sprint є готовий продукт (build), який можна передавати (deliver) замовнику (по крайней мере, система повинна бути готова до показу замовнику). Короткі спринти забезпечують швидкий feedback проектній команді від замовника. Замовник отримує можливість гнучко управляти scope системи, оцінюючи результат спринту і пропонуючи поліпшення до створеної функціональності. Ці вдосконалення потрапляють в Product Backlog, пріорітезіруются нарівні з іншими вимогами і можуть бути заплановані на наступний (Або на один з наступних) спринтів.Кожен спринт є маленький "водоспад". Протягом спринту робляться всі роботи по збору вимог, дизайну, кодування і тестування продукту.

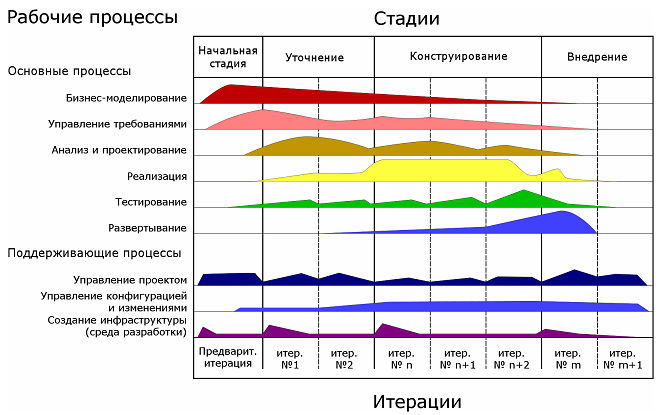
### **Difference between SCRUM & Waterfall**



* В Waterfall використовуюються довгі стадії, в Scrum - ітерації;
* В Waterfall планування і перегляд тест плану відбуваєтсья раз перед стартом девелопменту. В Scrum - кожного спринта;
* В Waterfall рекваєменти створюються раз на старті проекту. Вони сильно деталізовані і охоплюють весь скоуп та майже не змінюються. В Scrum рекваєменти генеруються і змінюються впродовж всього проекту;
* В Waterfall тест кейси створюються раз на старті проекту. В Scrum - кожного спринта;
* В Waterfall аксептанс тестінг проводиться кастомером після релізу. В Scrum - кожного спринта тест-командою, BA чи кастомером;
* В Waterfall дуже тяжко вносити зміни в заімплеменчені рішення, а Scrum є більш гнучким щодо цього;
* При використанні Scrum баги находяться набато раніше, оскільки при Waterfall тестування проводиться після завершення всієї фази розробки;
* З використанням Scrum набагто легше розробляти різні модулі аплікації і перевіряти їх спільну роботу на ранніх стадіях
* В Waterfall тестування починається лише після закінчення девелопменту і створення білда. В Scrum це відбувається паралельно;
* В Waterfall комунікація дев тіму з замовником відбувається дуже рідко і бюрократично. В Scrum команда може комунікувати з замовником кожного дня;
* В Waterfall естімейчення і асайнення проводиться лідом, кожен відповідає за свою маленьку частку функціоналу. В Scrum таски бере і естімейтить кожен сам, відповідальність за весь продукт;
* В Waterfall все регулює PM, в Scrum можуть працювати більш самодостатні команди.

### **RUP, Kanban, V-Model, XP, TDD**

**RUP**



Ітеративний фреймворк розробки програмного забезпечення. Він базується на будівельних блоках, які описують що повинно бути зробленим, необхідні навички та покрокове пояснення як досягати конкретні цілі. RUP використовує ітеративну модель розробки. В кінці кожної ітерації (в ідеалі триваючої від 2 до 6 тижнів) проектна команда повинна досягти запланованих на дану ітерацію цілей, створити або допрацювати проектні артефакти і отримати проміжну, але функціональну версію кінцевого продукту. Ітеративна розробка дозволяє швидко реагувати на мінливі вимоги, виявляти і усувати ризики на ранніх стадіях проекту, а також ефективно контролювати якість створюваного продукту. Є 4 фази життєвого циклу проекту:

* + - 1. Початкова фаза:

- оцінюється вартість і термін проекту;

- перевіряється чи зрозумілі вимоги;

2. Фаза уточення:

- створюється опис архітектури;

- фіналізуються рекваєменти;

- створюються прототипи;

3. Фаза конструювання:

- фаза імплементації продукту (девелопмент + тестінг);

- на великих проектах може бути кілька таких фаз;

4. Фаза впровадження:

- проводиться бета-тестування;

- тренінги по роботі з системою;

**Особливості RUP:**

* Всі фази проводяться на кожній ітерації;
* Вимоги можуть змінюватись в процесі розробки;
* Високий рівень формалізації і деталізації документів;

**Принципи**

В основі RUP лежать наступні принципи:

* Рання ідентифікація і безперервне (до закінчення проекту) усунення основних ризиків.
* Концентрація на виконанні вимог замовників до виконуваній програмі (аналіз і побудова моделі прецедентів (варіантів використання)).
* Очікування змін у вимогах, проектних рішеннях і реалізації в процесі розробки.
* Компонентна архітектура, реалізована і тестована на ранніх стадіях проекту.
* Постійне забезпечення якості на всіх етапах розробки проекту (продукту).
* Робота над проектом в згуртованій команді, ключова роль в якій належить архітекторам.

**Kanban**

Методологія, в основі якої лежить розробка функціоналу Just-in-time без перевантаження команди. Робота береться з черги. Весь процес є представлений візуально, де показано що, коли і в якій кількості повинно бути зроблено. В основі методології лежить впровадження інкрементальних змін в системі та **зменшення поточної кількості роботи**.

Відмінність від SCRUM:

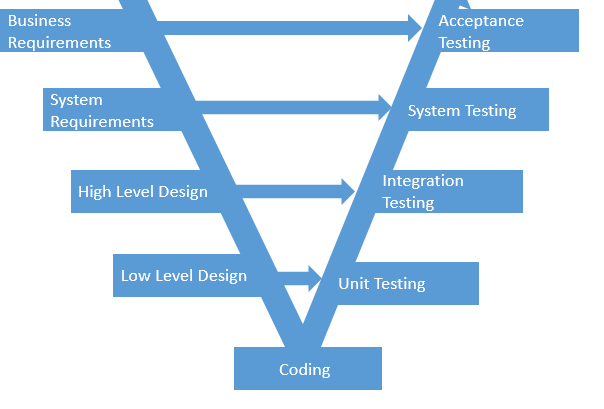
* Нема чітких таймбоксів для задач чи спринтів;
* Задачі більші і їх менше;
* Оцінки термінів завершення опціональні або відсутні;
* Прораховується лише середній час виконання завдання, а не швидкість команди.

Кожна стадія SDLCрозміщується на борді стовпцями. Над кожним стовпцем пишуться цифри, які позначають максимально можливу кількість карток в стовпці. Канбан добре показує ботлнеки (вузькі місця): якщо задевелопити можна 5 фіч за тиждень, а протестувати 3, то реалізовано буде лише 3 фічі, або більше, якщо девелопери допоможуть протестувати. В основі цієї методології лежить переключення між задачами, коли таски з якогось стовпця-стадії вже завершені. Ця методологія добре працює для супорту кількох проектів, проте не дуже добре підходить для проектів, де потрібно поєднувати тестування і розробку. В такому випадку краще ділити процес на стадію тестування і стадію розробки.

Загалом Канбан має 3 правила:

* Візуалізація розробки:
  + Робота ділиться на завдання, кожне завдання на доску;
  + На дошці пишуться назви стовпців, які відображають різні стадії розробки;
* Обмеження WorkInProgress на кожному етапі щоб побачити вузькі місця;
* Зміна часу середнього виконання завдання і оптимізація процесу для зменшення цього часу.

**V-Model**



Це розширення до Waterfall. Так само як і при Waterfall кожна наступна стадія розробки починається після закінчення попередньої. Проте тестування продукту планується паралельно з відповідною стадією розробки. Тобто на стадії:

- Requirement Analysis готуються аксептанс тести;

- SystemDesign і Architecture Design – план тестування системи;

- Module Design – дизайн інтеграційного і юніт тестування;

Цілі

V-модель забезпечує підтримку у плануванні та реалізації проекту. В ході проекту ставляться такі завдання:

* **Мінімізація ризиків:** V-подібна модель робить проект більш прозорим і підвищує якість контролю проекту шляхом стандартизації проміжних цілей і опису відповідних їм результатів та відповідальних осіб. Це дозволяє виявляти відхилення в проекті і ризики на ранніх стадіях і покращує якість управління проектом.
* **Підвищення та гарантії якості:** V-Model — стандартизована модель розробки, що дозволяє домогтися від проекту результатів бажаної якості. Проміжні результати можуть бути перевірені на ранніх стадіях. Універсальне документування полегшує читаність, зрозумілість та контрольованість.
* **Зменшення загальної вартості проекту:** Ресурси на розробку, виробництво, управління і підтримку можуть бути заздалегідь прораховані та проконтрольовані. Отримувані результати також універсальні і легко прогнозуються. Це зменшує витрати на подальші стадії та проекти.
* **Підвищення якості комунікації між учасниками проекту:** Універсальний опис усіх елементів та умов полегшує взаєморозуміння всіх учасників проекту. Таким чином, зменшуються неточності у розумінні між користувачем, покупцем, постачальником і розробником

### Переваги

* У моделі особливе значення надається плануванню, спрямованому на верифікацію та атестацію розроблювального продукту на ранніх стадіях його розробки. Фаза модульного тестування підтверджує правильність деталізованого проектування. Фази інтеграції та тестування реалізують архітектурне проектування або проектування на вищому рівні. Фаза тестування системи підтверджує правильність виконання етапу вимог до продукту і його специфікації.
* У моделі передбачені атестація та верифікація всіх зовнішніх і внутрішніх отриманих даних, а не тільки самого програмного продукту.
* У V-подібної моделі визначення вимог виконується перед розробкою проекту системи, а проектування ПО — перед розробкою компонентів.
* Модель визначає продукти, які повинні бути отримані в результаті процесу розробки, причому кожні отримані дані повинні піддаватися тестуванню.
* Завдяки моделі менеджери проекту можуть відслідковувати хід процесу розробки, так як в даному випадку цілком можливо скористатися тимчасовою шкалою, а завершення кожної фази є контрольною точкою

### Недоліки

* Модель не передбачає роботу з паралельними подіями.
* У моделі не передбачено внесення вимоги динамічних змін на різних етапах життєвого циклу.
* Тестування вимог в життєвому циклі відбувається занадто пізно, внаслідок чого неможливо внести змін, не вплинувши при цьому на графік виконання проекту.
* У модель не входять дії, спрямовані на аналіз ризиків.
* Деякий результат можна отримати тільки при досягненні низу букви V

**XP (Екстремальне програмування)**

Це одна з методологій Agile розробки, ціль якої підвищення рівня гнучкості системи щодо змін рекваєментів. Вона використовується переважно коли тяжко створити далекоглядні рекваєменти. Для того імплементуються поточні вимоги, які потім можна доволі легко змінити. Для того щоб ця методологія працювала потрібно:

* Постійно комунікувати з замовником і отримувати фітбек чи функціонал робиться правильно;
* Робити дизайн системи максимально простим;
* Впроваджувати TDD;
* Робити часті малі релізи;
* Налаштовувати CI;
* Найкраще працювати on-site;

Активності:

* Написання коду;
* Teстування;
* Слухання - уважно слухати потреби замовника і давати свій фітбек. Потім цей процес переростає у Planning Game;
* Дизайн - створення максимвльно простої архітектури системи.

XP має 12 рекомендованих практик:

* **Fine scale feedback**
  + Pair Programming - одночасна робота двох розробників за одним ПК з періодичною зміною ролей кодера і навігатора;
  + PlanningGame–практика, в основі якої лежить створення швидкого необробленого плану, уточнення якого відбувається протягом проекту. План включає оцінку рекваєментів, планування деліверання функціоналу, кінця проекту.
  + TestDrivenDevelopment - технологія розробки, яка передбачає попереднє написання тестів, поверх яких потім пишеться функціонал. Це дозволяє робити мене багів при написанні функціоналу.
  + WholeTeam - вся команда повинна тісно співпрацювати, включаючи замовника. Замовник в ідеалі повинен бути справжнім енд юзером програми.
* **Continuous process**
  + ContinuousIntegration - постійна робота на останній версії програми
  + DesignImprovement - так як імплементуєтья те що потрібно зараз, дизайн системи повинен покращуватись при розробці нових фіч.
  + SmallReleases - часті випуски програми, які дозволяють трекати реальний прогрес продукту.
* **Shared understanding**
  + CodingStandards - Визначення стандартів кодування всередині команди.
  + Collective Code Ownership - всі відповідають за весь код.
  + Simple Design - вибирається найпростіший дизайн, бо його легко менйтейнити і розширювати.
  + System Metaphor - коротка історія яка описує принцип роботи системи, яку знають всі мембери. З назв класів і методів, які матчаться в цю історію повинно бути зрозуміло, який функціонал вони імплементують.
* **Programmer welfare**
  + SustainablePace - робота не повинна займати більше 40 год\тиждень.

### Задачі

*Extreme Programming Explained* описує екстремальне програмування як дисципліну розробки програмного забезпечення яка змушує людей створювати високоякісне ПЗ якомога швидше.

ХР намагається зменшити ціну зміни вимог до ПЗ завдяки малим циклам розробки, а не одним довгим циклом. Екстремальне програмування сприймає зміни до вимог як звичайні, неминучі та бажані аспекти розробки ПЗ, і ці зміни мають бути очікуваним. Основна ідея полягає в тому що неможливо розробити самодостатній пакет вимог до ПЗ, зміни в вимогах - неминучі. Екстремальне програмування також вводить набір практик та принципів на основі методології [гнучкої розробки програмного забезпечення](https://uk.wikipedia.org/wiki/%D0%93%D0%BD%D1%83%D1%87%D0%BA%D0%B0_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F).

### Активності

Екстремальне програмування описує чотири базові активності що виконуються при розробці програмного забезпечення: написання коду, тестування, слухання та дизайн.

##### Написання коду

Прихильники ХР заявляють що єдиним дійсно важливим результатом розробки ПЗ є код: без готового коду нема продукту.

#### Тестування

Методологія екстремального програмування заявляє, що якщо дрібне тестування може перевірити незначну частину функціональності, то багато дрібних тестів можуть перевірити набагато більше частинок і продукт в цілому.

## Основні прийоми XP

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

* Короткий цикл зворотного зв'язку (Fine scale feedback)
  + [Розробка через тестування](https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D1%87%D0%B5%D1%80%D0%B5%D0%B7_%D1%82%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) (Test driven development)
  + Гра в планування (Planning game)
  + Замовник завжди поруч (Whole team, Onsite customer)
  + [Парне програмування](https://uk.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) (Pair programming)
* Безперервний, а не пакетний процес
  + [Безперервна інтеграція](https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D0%B7%D0%BF%D0%B5%D1%80%D0%B5%D1%80%D0%B2%D0%BD%D0%B0_%D1%96%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D1%96%D1%8F) (Continuous Integration)
  + [Рефакторинг](https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3) (Design Improvement, Refactor)
  + Часті невеликі релізи (Small Releases)
* Розуміння, що поділяється всіма учасниками
  + Простота (Simple design)
  + Метафора системи (System metaphor)
  + Колективне володіння кодом (Collective code ownership) або обраними шаблонами проектування (Collective patterns ownership)
  + [Стандарт кодування](https://uk.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82_%D0%BA%D0%BE%D0%B4%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F&action=edit&redlink=1) (Coding standard or Coding conventions)
* Соціальна захищеність програміста (Programmer welfare) :
  + 40-годинний робочий тиждень (Sustainable pace, Forty hour week)

### **Key activities and difference between phases**

### Покрито у Key phases of Fundamental Test process according to ISTQB standard

## Project Team Roles and Responsibilities

### **Understand typical Project Team structure and Project Team roles**

* Product owner - людина, яка представляє інтереси замовника. Він відповідає на запитання команди щодо вимог, визначає пріорітети юзер сторь.
* SRUM master - вирішення організаційних питань для дев-тіму. Від є драйвером на дейлі мітінгах, слідкує за порядком на дашборді і графіками.
* Development Team - люди, які безпосередньо роблять продукт.
* Де коли ще виділяють Stakeholders - люди, які зацікавлені проектом і мають до нього відношення (кінцеві користувачі, інвестори, менеджмент, супорт).

### Undestand typical QC Team roles and responsibilities

**Junior TA Engineer:**

1        Participates in the QC documentation preparation (test-cases/scenario, check lists, traceability matrixes, etc) based on the defined scope for test design and test design for regression testing taking into account changes in product requirements under supervision of QC Lead.

2        Performs the analysis, reporting, and tracking of defects/issues using a defined procedure and workflow.

3        Performs the analysis of defects related to regression testing and system changing from the defect fixing scope.

4        Executes the automated test tests and LOG analysis.

5        Writes the automated tests for functional and white-box testing using basic data retrieval and modifying algorithms, choosing appropriate common data containers and applying of basic OOP concepts.

**Junior QC Engineer:**

1        Executes all testing procedures to determine and guarantees required product quality. Understands QC activities and responsibilities on a project.

2        Tests software thoroughly using required test types at various stages of a project lifecycle to ensure proper operation and reports all found defects for fixing.

3        Prepares QC documentation (test-cases/scenario, check lists, traceability matrixes, etc) based on the defined scope for test design and performs regression testing test design taking into account changes in product requirements.

4        Performs analysis, reporting, and tracking of product features/enhancement using a defined procedure and workflow.

5        Performs analysis, reporting, and tracking of defects/issues using a defined procedure and workflow.

6        Performs analysis of defects, impacts analysis related to regression testing and system changes in the scope of defects fixing.

**TA Engineer:**

1        Executes automated tests scripts and performs log analysis.

2        Writes automated tests for functional, white-box and performance testing using basic data retrieval and modifying algorithms, choosing the appropriate common data containers and applying the OOP concepts.

3        Performs the automation test framework modules refactoring in the scope of the defined architecture under Senior staff supervision.

4        Performs impact analysis of all possible items inspecting specifications and requirements for the completeness and inaccuracies and changing requirements to other requirements/design/ test cases.

5        Performs analysis, reporting, and tracking of defects/issues using the defined procedure and workflow.

6        Verifies that defects, issues, and other items are completely resolved and that all appropriate items are updated accordingly.

**QC Engineer:**

1        Executes all quality control procedures to determine and guarantee required product quality. Fully understands QC activities and responsibilities on a project.

2        Perform testing of defined test procedure, execute the created test cases and log the test results

3        Specifies the testing approaches based on the selected items for testing and its features.

4        Performs the impact analysis for all possible inspecting items: specifications or requirements for the inaccuracies and completeness and changing requirements to other requirements/design/test cases.

5        Designs the tests: specifies the details of the test approach for software features and creates associated tests.

6        Performs the impact analysis related to regression testing and system changing from defect fixing scope.

7        Creates, updates, supports, and provides the review of specifications, requirements, or other documents related to QC activities on a project when needed.

8        Verifies that defects, issues, and other items are completely resolved and all appropriate items are updated accordingly.

**Senior TA Engineer:**

1        Plans the automation test framework components’ refactoring according to the product automated testing roadmap and the recognized issues.

2        Performs refactoring of the automation test framework modules in the scope of the defined architecture.

3        Analyzes the possible effectiveness of test automation. Defines and selects possible product areas for test automation.

4        Provides code inspections - reviews other’s code and provides valuable and comprehensive feedback to the code developer.

5        Determines the scope, strategy, approach, resources, test types and items that can be applicable and effective for the particular project and schedule of quality control on projects.

6        Performs the effective scheduling of all AQC activities within the project to ensure optimal AQC process during the SMC under QC/TA Lead supervision.

7        Performs analysis, reporting, and tracking of defects/issues using a defined procedure and workflow.

8        Performs impact analysis and inspects all possible items for completeness and inaccuracies based on: specifications and requirements, change requests, regression testing, defects’ fixing.

9        Provides coaching of other team members: assigns to the project, supports in all AQC activities whet it is needed, and provides review and consultation.

**Senior QC Engineer:**

1        Determines scope, strategy, approach, resources, test types, and items that can be applicable and effective for the particular project. Monitors and schedules quality control on projects

2        Performs the effective scheduling of all project QC activities to ensure the most effective QC process during the SMC.

3        Manages requirements changes and their influences on the product on the whole.

4        Defines possible or active project issues and risks, anticipates software release problems and takes corrective actions, escalates them, if needed, to resolve and achieve benefits.

5        Participates in setting up QC procedures on the project: defect tracking, managing of changing requirements, test design, testing, meetings, communication, etc

6        Ensures that tests and procedures are properly understood, carried out, and evaluated and that product modifications are investigated if necessary.

7        Performs analysis, reporting, and tracking of defects/issues using a defined procedure and workflow.

8        Performs impact analysis and inspection of all possible items for inaccuracies and completeness of realization based on specifications or requirements, changes to requirements, system changes related to regression testing or defect fixing; preparation of corresponding documents/conclusions to avoid similar problems in the future.

9        Identifies test design: defines scope, expediency, and types of test design and testing according to selected test types to ensure product quality. Finds and documents product discrepancies and works with other technical personnel on their fixing.

10    Provides coaching of other team members: assigns to the project, supports in all QC activities whet it is needed, and provides review and consultation.

**QC Leader:**

1        Implements best practices and methods for product testing.

2        Creates or coordinate creation of QC project documentation: Test Plan, Test Strategy, Test Design, etc.

3        Coordinates QC project activities with the customer within the project.

4        Plans QC team resource load, performs status tracking and retain stuff in assigned projects.

5        Researches project areas which can be improved and suggests improvements.

6        Suggests areas for improvements in company internal processes along with possible solutions.

7        Establishes policies and procedures that contribute to the efficiency.

8        Cooperates with clients, identifies their needs, and manages client requirements and expectations.

9        Provides peer review of documentation and suggests solid comments for improvements to the customer as well as to SoftServe staff.

10    Provides risks identification and prediction: existing and possible that can impact QC work and customer.

11    Plans and participates in technical meetings.

12    Provides coaching of other team members: assigns to the project, supports in all QC activities whet it is needed, and provides review and consultation.

13    Develops QC metrics for performance measuring on the team and individual levels. Monitors and measures performance through gathering relevant data and producing statistical reports, can evaluate the success of QC processes and improve it on the basis of gathered metrics.

**QA Manager:**

1        Works with Director of QA to oversees all quality assurance effort within the organization including development and implementation of Quality Approach

2        Jointly with Director of QA develops and implements quality approach within the projects according to industry standards

3        Works on establishing joined criteria for process quality reviews and accordingly to those evaluate and continuously improves QA process ensuring that the quality assurance and control processes meet organization and industry standards

4        Develop swith Director of QA one quality strategy for the company and tailor it for each particular project

5        Develops knowledge libraries for company internal use

6        Manages SoftServe QC resources, build and maintain SoftServe QC team structure for meeting successful project objectives

7        Coordinates with Director of QA regarding overall QC team structure that ensures quality level for the product line in order to use all resources available in the most efficient way. Performs high-level planning of work load for SoftServe QC team members, their load and performance, status tracking and retaining staff in assigned projects

8        Shares SoftServe’s best practices and methods for product testing (agile testing, automated testing, SaaS testing, etc.)

9        Monitors projects quality metrics to track quality for the product line

10    Manages and assign individual tasks and responsibilities to direct reports

11    Supervises software projects and provide ongoing support for QC staff

12    Establishes policies and procedures that contribute to the efficiency

13    Provides product quality risks management

### **Understand the difference between QC level and QC role**

Мабуть стосується розуміння обов’язків на різному рівні досвіжу QC інженера

**Quality Manager**

Лідерство та коучінг при підготовчих та виконавчих активностях, в тому числі:

* планування ресурсів4;
* комунікація між командами та їх статуси;
* нагляд за використанням бест практик;
* налагоддження зв’язку між замовником та командою.

**Quality Lead**

* відповідальний за результати тестування та використання практик;
* асайнення тасок;
* визначення рівня автоматизації.

**Tester(s)**

* Валідація рекваєментів;
* Дизайн та імплеменнтація тесів;
* Виконання тестів, репортинг і діагностування багів;
* Підртимка тестових процесів на високому рівні.

### Understand and explain cooperation and common process flow between roles in Project Team

### Understand and explain roles and responsibilities in distributed environment

Такі ж як і в інших командах, тільки скрам мастер повинен відповідати за якість зв’язку на мітингах.

### Understand difference between QC and QA roles

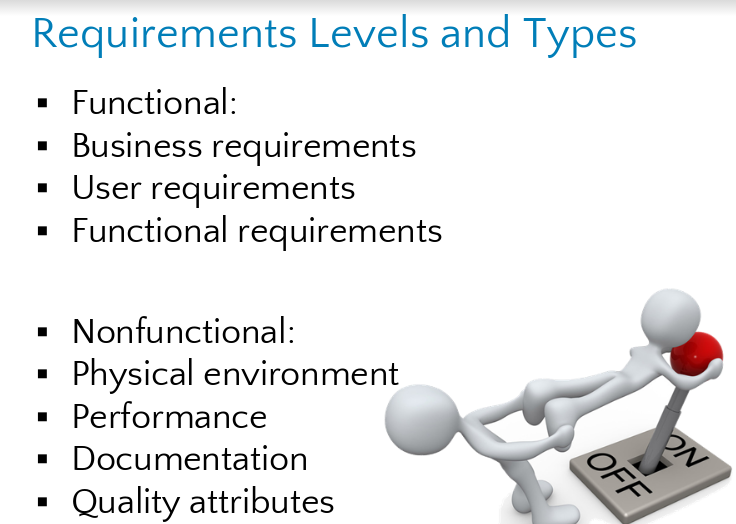
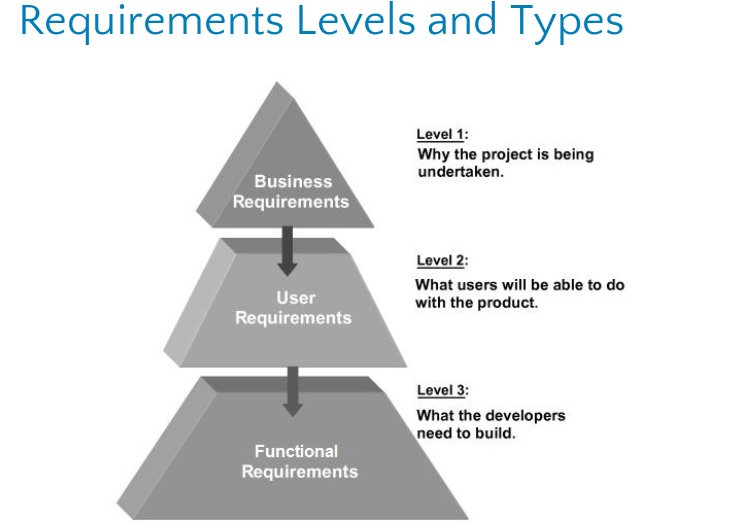
QC – тестує імплементований функціонал;

QA–намагаєтсья запровадити практики щоб попередити виникнення багів.

# **Working with requirements**

## Requirements Analysis

### **Types of documents through which requirements can be communicated**

1. Architectural Requirements - вимоги щодо архітектури системи.
2. UI Requirements - опис вимог до GUI і UX аплікації.
3. Documentation Requirements–вимоги щодо документів, які повинні бути заделіверані замовнику з аплікацією.
4. Functional Requirements - деталізоване описання роботи системи з конкретними кроками переходу з одного стану в інший.
5. Physical Requirements - вимогидо апаратного забезпечення, на якому повинна працювати аплікація.
6. User and human factor Requirements - вимогищодо системи для конкретних типів юзерів.
7. Performance&Load Requirements – вимоги до витримування навантажень системою.
8. Security Requirements – Вимоги щодо безпеки системи.

Types of documents:

* SRS– документ що містить всі вимоги до системи, згруповані за типом;
* Product Backlog – список юзер сторь.
* UseCases – вимоги, які описують якісь загальні флови до системи. Переважно зображаються у вигляді діаграм

**Вимоги споживача**

вирази фактів та припущень які описують очікування до системи в термінах цілей, середовища, обмежень, та міри ефективності й придатності. Споживачі це ті, хто виконують вісім первинних функцій системної інженерії, з особливим наголосом на операторі, як на ключовому споживачі. Операційні вимоги опишуть базову необхідність, і як мінімум дадуть відповідь на запитання, з даного списку:

* *Операційне поширення і розгортання*: Де використають систему?
* *Профіль чи сценарій місії*: Як система буде виконувати свої завдання?
* *Продуктивність та пов'язані параметри*: Які параметри критичні для виконання місії?
* *Використання середовища*: Як будуть використовуватись різноманітні компоненти системи?
* *Вимоги ефективності*: Якою ефективною має бути система для виконання своєї місії?
* *Операційний життєвий цикл*: Як довго система буде використовуватись споживачем?
* *Середовище*: Яких середовищ система очікує щоб працювати ефективно?

**Архітектурні вимоги**

Архітектурні вимоги пояснюють що має бути зроблено ідентифікацією необхідної [системної архітектури](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0).

**Структурні вимоги**

Структурні вимоги пояснюють що має бути зроблено ідентифікацією необхідної [структури](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0) системи.

**Поведінкові вимоги**

Поведінкові вимоги пояснюють що має бути зроблено ідентифікацією необхідної [поведінки](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D0%B5%D0%B4%D1%96%D0%BD%D0%BA%D0%B0) системи.

**Функціональні вимоги**

[Функціональні вимоги](https://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%96_%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3%D0%B8) пояснюють що має бути зроблено ідентифікацією необхідної задачі, дії, чи діяльності які мають виконуватись. Аналіз функціональних вимог буде використаний в функціях верхніх рівнів для функціонального аналізу.

**Нефункціональні вимоги**

[Нефункціональні вимоги](https://uk.wikipedia.org/wiki/%D0%9D%D0%B5%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%96_%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3%D0%B8) — це вимоги що задають критерій для оцінки операцій системи, замість її поведінки.

**Вимоги продуктивності**

До якої міри місії чи функції повинні бути виконані; зазвичай вимірюється в термінах кількості, якості, охопленні, своєчасності чи готовності. Протягом аналізу вимог, вимоги продуктивності (як добре воно має бути зроблено) будуть інтерактнивно розроблятись вздовж всіх виявлених функції що базуються на факторах життєвого циклу системи, і характеризуються в термінах ступеня визначеності в їх оцінках, ступеня критичності успіху системи, і їх відношення до інших вимог.

**Вимоги дизайну**

Вимоги «будувати до», «кодувати до», і «купувати до» для продуктів, і «як виконати» для процесів виражених в технічних пакетах даних та інструкціях.

**Успадковані вимоги**

Вимоги які маються на увазі вимогами вищого рівня, чи перетворені з них. Наприклад вимога великої дальності, чи високої швидкості може спричинити вимогу дизайну малої ваги.

**Розподілені вимоги**

Вимоги які визначені поділом, чи іншим перерозміщенням високорівневих вимог в кілька низькорівневих вимог. Наприклад стокілограмовий пристрій що складається з двох підсистем може спричинити вимоги ваги не більше 70 та 30 кілограм для конкретних систем нижчого рівня.

До відомих моделей категоризації вимог належать [FURPS](https://uk.wikipedia.org/w/index.php?title=FURPS&action=edit&redlink=1) та FURPS+, розроблені в [Hewlett-Packard](https://uk.wikipedia.org/wiki/Hewlett-Packard).

### **Purpose and importance of requirements review**

Рекваєменти потрібно переглядати щоб забезпечити правильну імплементацію функціоналу через:

* Коректність – опис функціоналу повинен метатись в загальну роботу системи;
* Показувати залежність – повинно бути чітко зрозуміло на який функціонал впливають рекваєменти;
* Повнота – чи є зрозумілий і повний опис;
* Можливість перевірити – повинні бути аксептанс критерії;
* Без двозначностей;

### **Importance and purpose of requirements traceability**

* Відслідковування залежностей і повноту покриття системи;
* Зрозуміти походження рекваємента і послідовність імплементації;
* Відслідковування змін і наслідків;

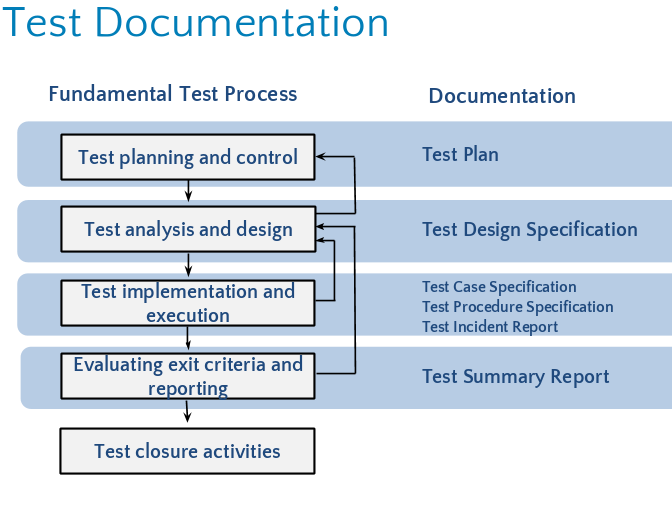
### **Purpose of traceability and dependency matrixes**

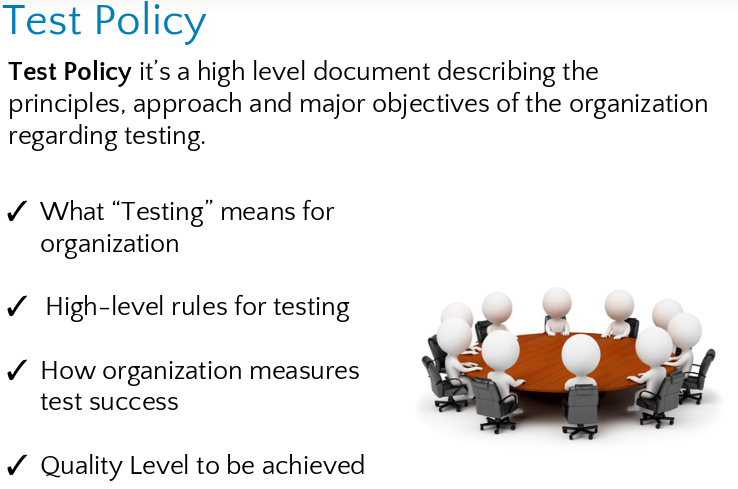
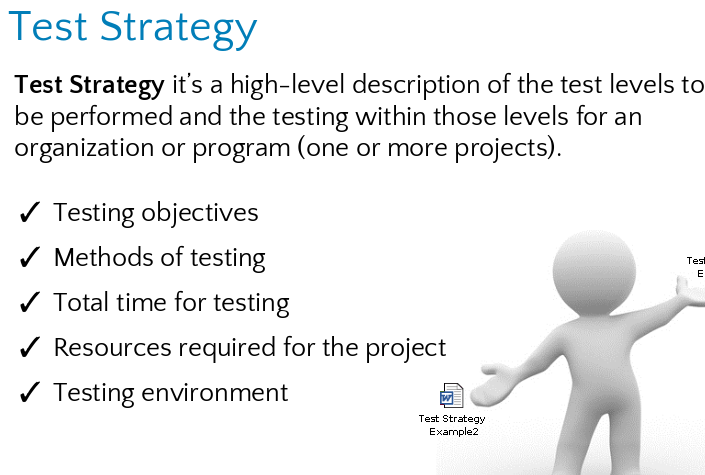
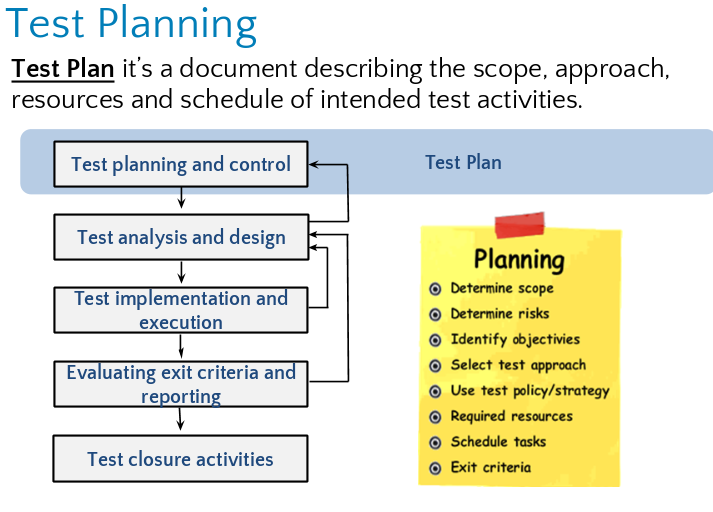
* Відслідковування залежностей і повноту покриття системи;
* Зрозуміти походження рекваємента і послідовність імплементації;
* Відслідковування змін і наслідків;
* Відслідковування покриття тестами через залежності між вимогами.

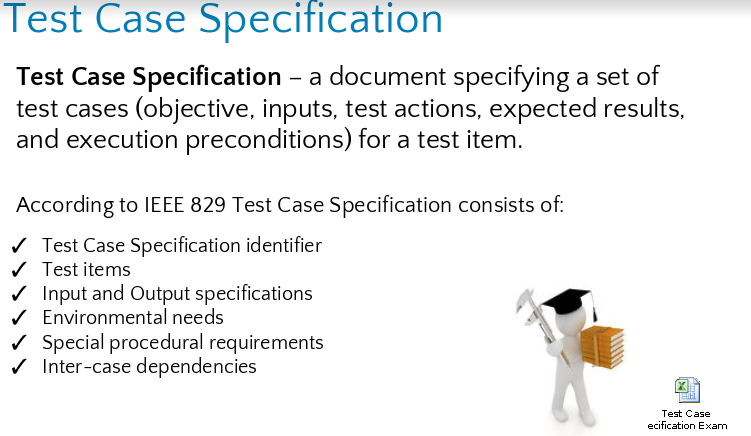
# **Test design**

## Test Development Process

### **Purpose and different types of test documentation**





**Test Plan**

* Test Plan identifier
* Introduction
* Test items
* Features to be tested
* Features not to be tested
* Approach
* Item pass/fail criteria
* Suspension criteria and resumption requirements
* Test deliverables
* Testing tasks
* Environmental needs
* Responsibilities
* Staffing and training needs
* Schedule
* Risks and contingencies
* Approvals

Тестова стратегія – загальний підхід до тестування та набору документів на виході.

Тест план – документ, який описує:

* що буде протестоване;
* якими методами;
* які терміни тестування;
* який буде рівень покриття тестами.

Тест дизайн специфікація – документ, який містить список фіч, які будуть тестуватись, а також список кейсів, які повинні бути протестовані.

Тест кейс специфікація – документ, який містить набір тест кейсів з детальним описом, описує їх залежності, вимоги до середовища.

Тест кейс – набір кроків і вхідних даних, який перевіряє відповідність певного флову системи відповідним рекваєментам.

### **Specification-based (black-box) approaches to test design**

Black-box тестування базується на перевірці роботи системи без докладного знання коду і архітектури, а лише з використанням рекваєментів чи усних розмов з замовником якщо описаних рекваєментів нема.Такий тест дизайн будується з використанням різних технік тестування.

### **Main black-box test design techniques**

Specification-Based Test Design Techniques (black-box test design technique): Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.

**Specification-based techniques include:**

**Goals:**

1. To reduce the number of test cases to necessary minimum
2. To select the right test cases to cover all possible scenarios
   * Equivavence partitioning
   * Boundary value analysis
   * Decision tables
   * State transition testing
   * Use case testing

Є такі основні техніки:

* Класи еквівалентності – групування тестових сценаріїв, що приводять до однакового результату;
* Тестування граничних значень – тестування з допомогою граничних даних на певних проміжках;
* DecisionTable – тестування різних воркфловів з допомогою таблиці, яка містить набір вхідних умов (дані чи конфігурація) та бізнес правил, які переходять в дію та очікуваний результат;
* StateTransition – тестування переходів системи між різними станами;
* UseCasetesting – тестування воркфловів залежно від типу юзера;
* MindMap – описання складових якоїсь одної ідеї.

# **Test Execution and Defects Management**

## Test Execution

### **Test Levels and Test Types**

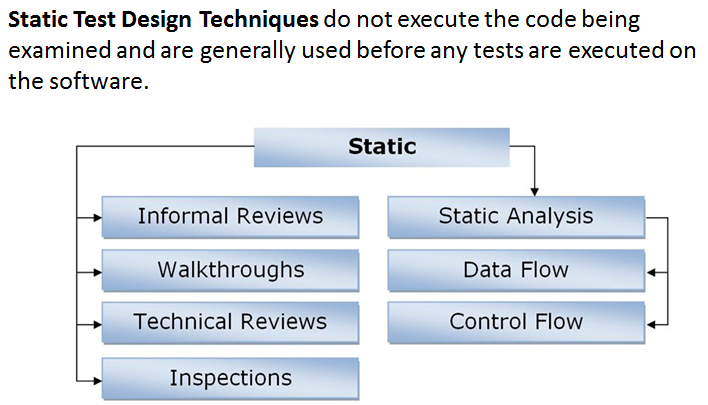
Test Levels:

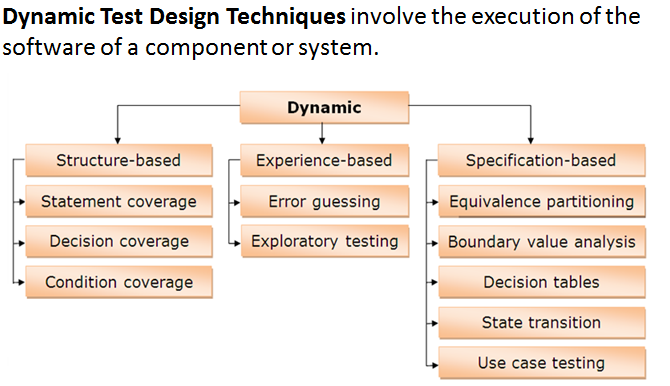
* Component (Unit) testing – тестування якоїсь функції в коді, часто використовують заглушки при їх розробці. Виконує девелопер.
* Integration testing – перевірка роботи системи при взаємодії кількох компонент і перевірка роботи аплікації без заглушок, а використовують реальні робочі сервіси. Виконує девелопер і тестувальник.
* System testing – тестування цілої системи на її відповідність поставленим рекваєментам. Проводиться тестувальником.
* Acceptance testing – тестування системи людиною зі сторони замовника на відповідність аксептанс критеріям.

Test Types:

* Smoke testing – набір тестових сценаріїв, які перевіряють основні воркфлови функціональності чи системи;
* Sanity testing – сценарій, який перевіряє чи результат проходження кейсу потенційно є позитивним;
* GUI testing – перевірка графічного інтерфейсу програми на відповідність стандартам;
* Functional testing - перевірка виконання системою функцій, які передбачені в рекваєментах;
* Load testing - перевірка стабільності роботи програми при одночасній роботі певної кількості користувачів;
* Performance testing - перевірка стабільності роботи програми при виконанні якоїсь кількосіті операцій за одиницю часу;
* Confirmation testing - перевірка програми після фіксу баги;
* Regression testing - перевірка пов\*язаного функціоналу системи, який міг бути зачеплений при фіксі баги. Метою цього тестування є перевірка чи фікс не спричинив нових багів в функціоналі;

### **Difference between static and dynamic testing**





**Static testing techniques**

rely on the manual examination (reviews) and automated analysis (static analysis) of the code or other project documentation

**Reviews**

are a way of testing software work products (including code) and can be performed before dynamic test execution

**Dynamic**

Specification-Based Test Design Techniques (black-box test design technique): Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.

1. При статичному тестуванні код програми не виконується, а просто рев\*ювається, перевіряється документація.
2. Метою статичного тестування є передбачення дефектів на ранніх стадіях розробки продукту., а динамічного - пошук вже зроблених дефектів.
3. Статичне тестування супроводжується створенняс чекліста, а динамічне - тест кейсами.
4. Статичне виконуєтсья перед чекіном і білдом
5. Статичне тестування дешевше.

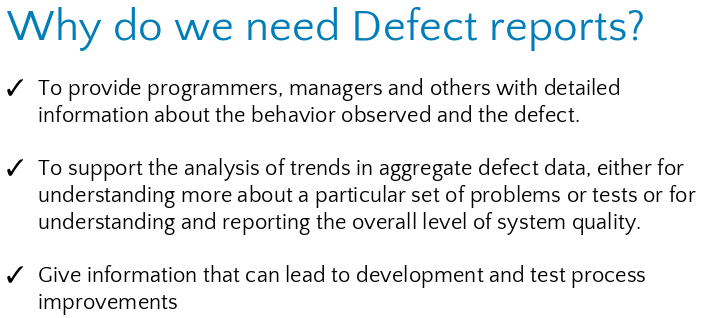
### **Types of test execution reports**

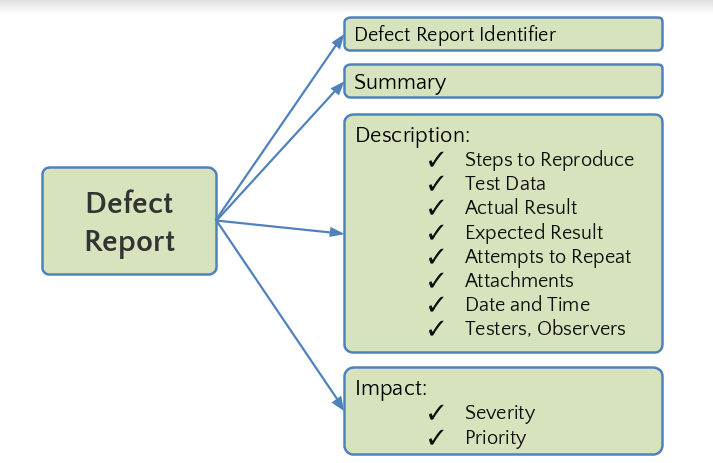
* Статус тест плану - поточний стан "Тест План" документу: завершений, був апдетйтнутий;
* Статус тест документації - поточний стан тест кейс документації;
* Статус виконання тесту - кількість і стан виконання запланованих тест кейсів, стан нових і поточних багів, енваєментів.

## Defects Tracking

### **What are defects reports for and how to write good ones**

DefectReport– документ, який містить інформацію про баг для його відтворення і фіксу. Він повинен містити:





* Унікальний номер;
* Назву, з якої максимально зрозуміло в чому проблема;
* Пріоритет;
* Поточний статус;
* Кроки відтворення;
* Очікуваний результат;
* Поточний результат;
* Інформація про середовище для відтворення.

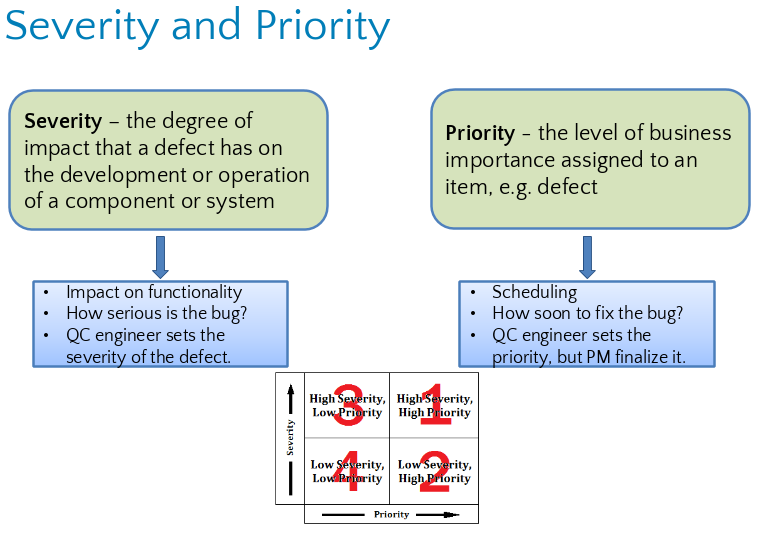
Tips & Tricks:

* Баг потрібно заводити відразу як зрозуміло що це баг, не відкладати;
* Потрібно максимально глибоко зрозуміти причину бага;
* Репорт повинен бути написаний простою мовою;
* Використання дженерік степів без зайвої конкретики;
* Потрібно давати максимально наочної інформації: скріншоти, логи, ерор меседжі і т.д… З тої інформації і назви дефекту повинно бути зрозуміло в чому проблема;

### **Severity and Priority**

Severity – ступінь впливу дефекту на роботу системи. Наприклад, якщо при кліканні на зовнішнє посилання система крешає, то це високе северіті.

Priority - серйозність багу з точки зору бізнесу, тобто як швидко він повинен бути пофіксаний. Наприклад помилка в хідері на головній сторінці це високий пріоритет.



HIGH SEVERITY – HIGH PRIORITY BUG

This is when major path through the application is broken, for example, on an eCommerce website, every customers get error message on the booking form and cannot place orders, or the product page throws a Error 500 response.

HIGH SEVERITY – LOW PRIORITY BUG

This happens when the bug causes major problems, but it only happens in very rare conditions or situations, for example, customers who use very old browsers cannot continue with their purchase of a product. Because the number of customers with very old browsers is very low, it is not a high priority to fix the issue.

HIGH PRIORITY – LOW SEVERITY BUG

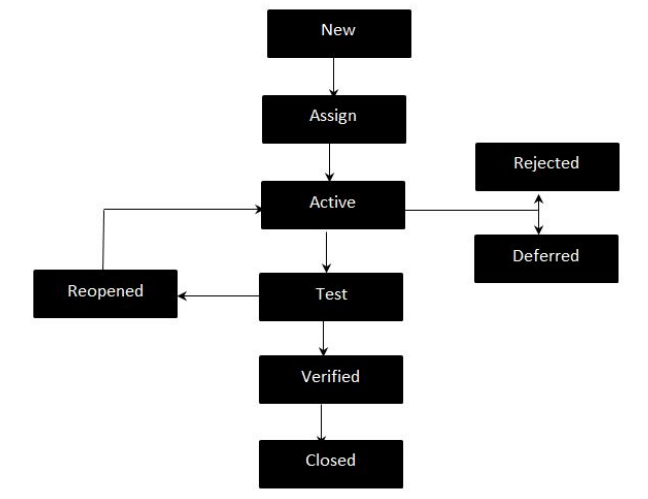
This could happen when, for example, the logo or name of the company is not displayed on the website. It is important to fix the issue as soon as possible, although it may not cause a lot of damage.

LOW PRIORITY – LOW SEVERITY BUG

For cases where the bug doesn’t cause disaster and only affects very small number of customers, both Severity and Priority are assigned low, for example, the privacy policy page take a long time to load. Not many people view the privacy policy page and slow loading doesn’t affect the customers much.

### **Defects life cycle**

Цестадії, якіпроходитьбагвпродовжперіодуіснування. Основні:

* Open
* In progress
* Fixed
* Closed
* Reopened
* 

# **Test Management**

## Risk Management

### **What is risk**

Це потенційна втрата якоїсь цінності об'єкту.

### **Difference between Risk, Issue and Problem**

**Issue** - це подія, яка вже відбулась і відоме її рішення. Воно має менш негативне значення, ніж problem.

**Risk** - подія, яка може потенційно відбутись.

**Problem** - ситуація, яке вже 100 % існує, але її вирішення потребує додаткового дослідження.

### **Difference between Project, Product and Business risks**

**Produc trisk** – це ерії в програмному коді, де існує найбільша кількість критичних багів. За цю ерію найбільш відповідальні тестувальники.

**Project risk** – це події, настання яких може призвести до подовження терміну проекту, причини яких можуть бути як зовнішні так і внутрішні. За цю ерію найбільш відповідальні прожект менеджери.

**Business risk** – це ризики всього бізнесу, такі як зміна законодавства, неврожай і т.д, причиною яких є зовнішні чинники.

### **Difference between Preventable, Strategy and External risks**

**Preventable risk** – внутрішні ризики, які є контрольовані і повинні бути усунуті. Це є група дефектів, які можуть існувати і фіксання яких є надто дорогим.

**Stategy risk** – це ризик, при якому буде отримано великі бенефіти або зазнається фіаско, оскільки це певний довгостроковий ризик, на кону якого багато стоїть.

Externalrisk – це зовнішні чинники, на які немає, переважно, впливу. Їх потрібно намагатись передбачати раніше і уникати.

### **Difference between risk probability and impact**

**Risk probability** – це ймовірність настання якоїсь події. Вона завжди є більше 0% і менше 100%, оскільки це б суперечило самому визначенню ризику.

**Risk impact** – це є наслідки настання певної події.

### **Identification of Causes and Effects**

Ризики і наслідки повинні бути ідентифіковані, через:

* правильне планування термінів з врахуванням ризиків;
* правильне планування бюджету;
* точніші рекваєменти;
* готовність команди до вирішення ішусів.

Це ітеративний процес.

### **Purpose of Risk Management**

Основною метою є визначення потенційних ризиків, їх аналіз та прийняття мір для їх уникнення. Це постійний процес. Ранній і агресивний ризик-менеджмент проводиться з залученням відповідних стейкхолдерів.

### **Main options that may be used to control the Risk**

1. Вилучити ризикові фактори, якщо бенефіси менші від наслідків;
2. Заміна ризикових факторів на менш ризикові;
3. Ізолювання ризику, наприклад створення захищених приміщень для зберігання токсичних речовин;
4. Автоматизація процесів де це доцільно для зменшення людського фактору;
5. Мінімізація ефекту ризику;
6. Реорганізація процесів для зменшення ймовірності настання ризику;
7. Вживання запобіжних заходів, таких як обмеження доступу до приміщень і т.д.;
8. Проведення тренінгів і запровадження менторства більш досвідченими колегами;
9. Використання засобів індивідуальної безпеки.

### **Risk-based Testing**

Це тип тестування, який базується на пріоритезації фіч і тест кейсів по ризику виникнення бага. Для проведення цього тестування потрібно:

* Скласти список фіч і кейсів по пріоритету;
* Провести тестування згідно списку;
* При виникненні нових ризиків притримуватись визначеного плану;

## Test Progress Monitoring and Control

### **Why metrics are necessary**

Метрики – це показники, які показують характеристики процесу.

Метрики важливі, бо вони допомагають:

* Допомагають зрозуміти поточний прогрес;
* Побачити чи досягаються поставлені цілі;
* Визначити проблемні місця та вжити відповідних заходів;
* Допомагаютьвизначити ризики та приймати рішення;

Метрики повинні бути легкими для пояснення та розуміння для того щоб всі розуміли їх потребу.

### **Common metrics used for monitoring test preparation and execution**

* Загальна кількість тест кейсів;
* Кількість виконаних тест кейсів;
* Кількість автоматизованих тест кейсів;
* Кількість пройдених\зафейлених тест кейсів;
* Загальний час танення тестів;
* Кількість запланованих і покритих фіч;
* Кількість відкритих дефектів з різними пріоритетами.

## Standards and Test Process Improvement

### **Sources of software standards and their usefulness for software testing**

Головні інститути, стандарти яких використовують в роботі:

* *InternationalOrganizationforStandardization, ISO* – міжнародна організація зі стандартизації, метою якої є ратифікація стандартів, розроблених спільними зусиллями делегатів з різних країн (**ISO/IEC 9126, ISO/IEC 25000:2005etc.**).
* *InstituteofElectricalandElectronicsEngineers, IEEE –* інститут інженерів з електротехніки та електроніки, який є світовим лідером в галузі розроблення [стандартів](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82) з [електроніки](https://uk.wikipedia.org/wiki/%D0%95%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D1%96%D0%BA%D0%B0) та [електротехніки](https://uk.wikipedia.org/wiki/%D0%95%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D1%82%D0%B5%D1%85%D0%BD%D1%96%D0%BA%D0%B0) (**IEEE 829-2008**, також відомий як **829 StandardforSoftwareandSystemTestDocumentation**).
* *BritishStandardsInstitute –*організація з розробки стандартів Великої Британії, яка стала основою для стандартів ISO (**BS 7925-1, BS 7925-2**).
* *InternationalSoftwareTestingQualificationsBoard, ISTQB –*незалежна організація, яка визначає основні знання, які повинні мати тестувальники у всьому світі (**ISTQB сертифікація**).

Головна цінність цих стандартів є використання одних і тих же термінів різними людьми та визначення загальних підходів до процесів, які є результатом багатьох років досвіду спеціалістів з різних країн.

# **Engineering Management**

## Process Planning (SDLC)

### **Softwaredevelopment process**

Це розбитий на окремі фази процес створення програмного продукту з використанням певної методології.

Процес розробки складається з безлічі підпроцесів, або дисциплін, деякі з яких показані нижче. У [моделі водоспаду](https://uk.wikipedia.org/wiki/%D0%92%D0%BE%D0%B4%D0%BE%D1%81%D0%BF%D0%B0%D0%B4%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) вони йдуть одна за одною, в інших аналогічних процесах їх порядок або склад змінюється.

* [Аналіз вимог](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D1%96%D0%B7_%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3) → [Специфікація програмного забезпечення](https://uk.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%86%D1%96%D1%8F_%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3_%D0%B4%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F)
* [Проектування програмного забезпечення](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F)
* [Програмування](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)
* [Тестування програмного забезпечення](https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F)
* [Системна інтеграція](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D0%B0_%D1%96%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D1%96%D1%8F)
* Впровадження програмного забезпечення (або [Установка програмного забезпечення](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%81%D1%82%D0%B0%D0%BB%D1%8F%D1%86%D1%96%D1%8F_%D0%9F%D0%97))
* [Супровід програмного забезпечення](https://uk.wikipedia.org/wiki/%D0%A1%D1%83%D0%BF%D1%80%D0%BE%D0%B2%D1%96%D0%B4_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F)

### **Software development activities**

* **Test planning**:
  + створення тестової стратегії і політики, врахування ризиків, визначення цілей тестування;
  + визначаються техніки тестування, рівень покриття, визначення deliverables для замовника, кількість людей та середовищ для тестування;
  + визначення definition of done;
  + потрібно впродовж всього тестування порівнювати план з поточним станом справ і інформувати керівників у випадку відхилень;
  + Оцінка результатів тестування та складання репортів.
* **Test analysisand Test design:**
  + Створоюються матреальні артефакти тестування: тест дизайни
  + Рев\*ювається специфікація і технічна документація
* **Test inmplementation and execution**
  + Написання тест кейсів, їх пріоретизація, створення тестових даних
  + Налаштовуються тестові середовища
  + Виконання тест кейсів
* **Evaluating exit criteria and reporting**
  + Перевірка тестових логів
  + Логування дефектів
  + Написання підсумкових репортів
* **Test closure activities**
  + Перевірка чи всі дефекти були залоговані
  + Чи всі deliverables були заделіверані)
  + Фіналізація тестових артефактів для майбутнього використання
  + Аналіз процесів для покращення їх в майбутньому

### Planning

Planning is an objective of each and every activity, where we want to discover things that belong to the project. An important task in creating a software program is extracting the [requirements](https://en.wikipedia.org/wiki/Requirement) or [requirements analysis](https://en.wikipedia.org/wiki/Requirements_analysis).[[5]](https://en.wikipedia.org/wiki/Software_development#cite_note-5) Customers typically have an abstract idea of what they want as an end result but do not know what *software* should do. Skilled and experienced software engineers recognize incomplete, ambiguous, or even contradictory requirements at this point. Frequently demonstrating live code may help reduce the risk that the requirements are incorrect.

Once the general requirements are gathered from the client, an analysis of the scope of the development should be determined and clearly stated. This is often called a scope document.

Certain functionality may be out of scope of the project as a function of cost or as a result of unclear requirements at the start of development. If the development is done externally, this document can be considered a legal document so that if there are ever disputes, any ambiguity of what was promised to the client can be clarified.

### Designing

*Main articles:*[*Software design*](https://en.wikipedia.org/wiki/Software_design)*and*[*Systems design*](https://en.wikipedia.org/wiki/Systems_design)

Once the requirements are established, the design of the software can be established in a [software design document](https://en.wikipedia.org/wiki/Software_design_document). This involves a preliminary, or [high-level design](https://en.wikipedia.org/wiki/High-level_design) of the main modules with an overall picture (such as a [block diagram](https://en.wikipedia.org/wiki/Block_diagram)) of how the parts fit together. The language, operating system, and hardware components should all be known at this time. Then a detailed or low-level design is created, perhaps with [prototyping](https://en.wikipedia.org/wiki/Software_prototyping) as proof-of-concept or to firm up requirements.

### Implementation, testing and documenting

[Implementation](https://en.wikipedia.org/wiki/Implementation) is the part of the process where [software engineers](https://en.wikipedia.org/wiki/Software_engineering) actually [program](https://en.wikipedia.org/wiki/Computer_programming) the code for the project.

[Software testing](https://en.wikipedia.org/wiki/Software_testing) is an integral and important phase of the software development process. This part of the process ensures that [defects](https://en.wikipedia.org/wiki/Software_bug) are recognized as soon as possible. In some processes, generally known as [test-driven development](https://en.wikipedia.org/wiki/Test-driven_development), tests may be developed just before implementation and serve as a guide for the implementation's correctness.

[Documenting](https://en.wikipedia.org/wiki/Software_documentation) the internal design of software for the purpose of future maintenance and enhancement is done throughout development. This may also include the writing of an [API](https://en.wikipedia.org/wiki/API), be it external or internal. The software engineering process chosen by the developing team will determine how much internal documentation (if any) is necessary. Plan-driven models (e.g., [Waterfall](https://en.wikipedia.org/wiki/Waterfall_model)) generally produce more documentation than [Agile](https://en.wikipedia.org/wiki/Agile_software_development) models.

### Deployment and maintenance

[Deployment](https://en.wikipedia.org/wiki/Software_deployment) starts directly after the code is appropriately tested, approved for [release](https://en.wikipedia.org/wiki/Software_release), and sold or otherwise distributed into a production environment. This may involve installation, customization (such as by setting parameters to the customer's values), testing, and possibly an extended period of evaluation.[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

Software training and [support](https://en.wikipedia.org/wiki/Technical_support) is important, as software is only effective if it is used correctly.[[6]](https://en.wikipedia.org/wiki/Software_development#cite_note-6) .[[7]](https://en.wikipedia.org/wiki/Software_development#cite_note-7)

[Maintaining](https://en.wikipedia.org/wiki/Software_maintenance) and enhancing software to cope with newly discovered [faults](https://en.wikipedia.org/wiki/Software_bug) or requirements can take substantial time and effort, as missed requirements may force redesign of the software.

### Other

* [Performance engineering](https://en.wikipedia.org/wiki/Performance_engineering)

Software Development is estimated to be a vastly expanding area. With how rapidly technology is changing, this skill has become more and more needed.

### **Abiliton SDLC**

Це адаптивнийфреймворк кращих практик для людей та процесів в межах SDLC. Він супроводжується постійними вимірами метрик та покращенням. Середнє збільшення ефективності процесів = 30% за 6 місяців

**Waterfall Concept**

Водоспадна модель життєвого циклу ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *waterfall model*) була запропонована в 1970 р. Вінстоном Ройсом. Вона передбачає послідовне виконання всіх етапів проекту в строго фіксованому порядку. Перехід на наступний етап означає повне завершення робіт на попередньому етапі. Вимоги, визначені на стадії формування вимог, суворо документуються у вигляді технічного завдання і фіксуються на весь час розробки проекту. Кожна стадія завершується випуском повного комплекту документації, достатньої для того, щоб розробка могла бути продовжена іншою командою розробників.

Етапи проекту у відповідності з каскадною моделлю:

1. Формування вимог;
2. Проектування;
3. Реалізація;
4. Тестування;
5. Впровадження;
6. Експлуатація та супровід.

**Переваги**

* Повна і погоджена документація на кожному етапі;
* Легко визначити терміни і витрати на проект.

**Недоліки**

У водоспадної моделі перехід від однієї фази проекту до іншого передбачає повну коректність результату (виходу) попередньої фази. Однак неточність будь-якої вимоги або некоректна його інтерпретація в результаті призводить до того, що доводиться «відкочуватися» до ранньої фазі проекту і необхідна переробка не просто вибиває проектну команду з графіка, але часто призводить до якісного зростання витрат і, не виключено, до припинення проекту в тій формі, в якій він спочатку замислювався. На думку сучасних фахівців, основна помилка авторів водоспадної моделі полягає у припущеннях, що проект проходить через весь процес один раз, спроектована архітектура хороша і проста у використанні, проект здійснення розумний, а помилки в реалізації легко усуваються в міру тестування. Ця модель виходить з того, що всі помилки будуть зосереджені на реалізації, а тому їх усунення відбувається рівномірно під час тестування компонентів і системи[[1]](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F#cite_note-1). Таким чином, водоспадна модель для великих проектів мало реалістична і може бути ефективно використана тільки для створення невеликих систем

**Agile software development concept**

Agile — родина процесів розробки, а не єдиний підхід в розробці програмного забезпечення, і визначається Agile Manifesto. Agile не включає практик, а визначає цінності та принципи, якими керуються успішні команди.

Agile Manifesto розроблений і прийнятий 17 розробниками 11-13 лютого 2001 року на лижному курорті The Lodge at Snowbird в горах Юти. Маніфест підписали представники наступних методологій [Extreme programming](https://uk.wikipedia.org/wiki/%D0%95%D0%BA%D1%81%D1%82%D1%80%D0%B5%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), [Scrum](https://uk.wikipedia.org/wiki/Scrum), [DSDM](https://uk.wikipedia.org/wiki/DSDM), [Adaptive software development](https://uk.wikipedia.org/wiki/%D0%90%D0%B4%D0%B0%D0%BF%D1%82%D0%B8%D0%B2%D0%BD%D0%B0_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0), Crystal Clear, [Feature driven development](https://uk.wikipedia.org/wiki/Feature_Driven_Development), Pragmatic Programming. Agile Manifesto містить 4 основні ідеї та 12 принципів. Примітно, що Agile Manifesto не містить практичних порад.

**Основні ідеї**:

* Особистості та їхні взаємодії важливіші, ніж процеси та інструменти;
* Робоче програмне забезпечення важливіше, ніж повна документація;
* Співпраця із замовником важливіша, ніж контрактні зобов'язання;
* Реакція на зміни важливіша, ніж дотримання плану.

**Принципи, які роз'яснює Agile Manifesto:**

* задоволення клієнта за рахунок ранньої та безперебійної поставки коштовного програмного забезпечення;
* вітання змін вимог навіть наприкінці розробки (це може підвищити конкурентоспроможність отриманого продукту);
* часта поставка робочого програмного забезпечення (кожен місяць або тиждень або ще частіше);
* тісне, щоденне спілкування замовника з розробниками впродовж всього проекту;
* проектом займаються мотивовані особистості, які забезпечені потрібними умовами роботи, підтримкою і довірою;
* рекомендований метод передачі інформації — особиста розмова (віч-на-віч);
* робоче програмне забезпечення — найкращий вимірювач прогресу;
* спонсори, розробники та користувачі повинні мати можливість підтримувати постійний темп на невизначений термін;
* постійну увагу поліпшенню технічної майстерності та зручному дизайну;
* простота — мистецтво не робити зайвої роботи;
* найкращі технічні вимоги, дизайн та архітектура виходять у самоорганізованої команди;
* постійна адаптація до мінливих обставин.

Маніфест та Принципи гнучкої розробки містять високорівневі ідеї щодо того, як потрібно вибудовувати процес розробки програмного забезпечення, щоб успішно завершувати проекти й створювати команди, в яких приємно та цікаво працювати. Документи визначають, що потрібно для цього зробити, але не говорять, як це зробити. По-іншому й не могло бути, оскільки Маніфест та Принципи народилися внаслідок консенсусу представників різних (хоча й споріднених) напрямів, які могли знайти спільну основу лише на рівні базових цінностей та принципів.

### **Evaluating technical feasibility**

Оцінка проекту:

* Оцінка proof of concept;
* Вибір кращих технологій для реалізації проекту;
* Зважування міркувань імплементації;
* Дослідження подібих рішень для бізнес-потреби.

### **Defining unit-testing process**

Одніює з головних функцій юніт тестів є перевірка чи зміна в коді не поламала поточний та пов’язаний функціонал. Що покривати тестами допомагають зрозуміти техніки тест-дизайну. Легкість покриття тестами повинна бути закладена в архітектурі самого коду.

### **Defining code review process**

Юніт тести не є єдиним ефективним способом перевірки коду. Існують баги в специфікації, в припущеннях, брак досвіду. Тому завжди потібні сторонні люди, які можуть переглянути код і підказати на помилки.

Найбільш популярнимим є такі техніки рев’ю:

* Inspection – пошук дефектів рев’ювером в коді з допомогою чек-ліста тестових сценаріїв;
* CodeWalkthroughs – автор коду розказує логіку коду, а інші задають запитання;
* PairProgramming – програмування разом з кимось;

Протягом рев’ю потрібно перевіряти на ступні пункти:

* Дотримання дизайну;
* Coding стандарти;
* Підтримуваність коду;
* Наявність документації;
* Безпека;
* Швидкодія.

Поради для якісного рев’ю:

* Перевіряти за один підхід не більше 400 стрічок коду;
* Швидкість рев’ю не повинна бути 400 стрічок за годину;
* Один підхід перевірки не повинен перевищувати 90 хвилин;
* Перед рев’ю код повинен бути покоментованим;
* Процес рев’ю повинен мати кількісно виражені цілі та супроводжуватись метриками;
* Створювати чеклісти що потірбно перевірити;
* Перевіряти що баги є справді пофіксані на UI;
* Знайдені баги під час рев’ю є позитивним моментом;
* Потрібно остерігатись “BigBrother” ефекту. Метрики повинні використовуватись виключно для покращення процесів;
* Якщо нема часу на повне рев’ю, то потріюно перевірити хоча б частину;
* Рев’ювати код потрібно частіше і меншими порціями.

## Estimation

### Scope Concept

In project management, the term scope has two distinct uses- Project Scope and Product Scope.

Project Scope - "The work that needs to be accomplished to deliver a product, service, or result with the specified features and functions."

Product Scope - "The features and functions that characterize a product, service, or result."

Notice that Project Scope is more work-oriented, (the hows,) while Product Scope is more oriented toward functional requirements. (the whats.)

### **Estimates, Targets, and Commitments**

Estimate is correct: an estimate is a prediction of how long a project will take or how much it will cost. But estimation on software projects interplays with business targets, commitments, and control. A target is a statement of a desirable business objective. Businesses have important reasons to establish targets independent of software estimates. But the fact that a target is desirable or even mandatory does not necessarily mean that it is achievable.While a target is a description of a desirable business objective, a commitment is a promise to deliver defined functionality at a specific level of quality by a certain date. A commitment can be the same as the estimate, or it can be more aggressive or more conservative than the estimate. In other words, do not assume that the commitment has to be the same as the estimate; it doesn’t.

### **Overestimate vs Underestimate**

Наслідки недоестімейтів:

* Набір меншої кількості людей, ніж потрібно;
* Люди, як правило недоестімейчують 30% роботи. Тому навіть нормальний естімейт це ризик не здати проект вчасно;
* Малі затрати часу на рекваєменти і архітекртуру, в результаті чого потім прийдеться переробляти;
* Команда під кінець проекту втягується в непотрібні активності “Як встигнути здати проект”;

Наслідок переестімейту:

* Спрацьовує закон Паркінсона – робота займає весь виділений на неї час.

Моя думка така, що краще переестімейтити.

Головне правило: чим точніші рекваєменти тим точніший естімейт.

### **DecompositionandRecomposition**

Декомпозиція – розбиття на частини якосго цілого, окремий естімейт для кожного та злиття всіх естімейтів до купи. Тут діє закон великих чисел: якщо робити естімейт чогось одного, то похибка робиться 1 раз і в одну сторону (забагато або замало). Якщо естімейтити окремі частини, то частина похибок менша реальної оцінки, а частина – більше. При сумуванні відбувається вирівнювання загальної оцінки.

Знаходження загального середнього відхилення при розбивці на задачі:

* Знайти дисперсію для кожної задачі;
* Просумувати дисперсії;
* Знайти квадратний корінь з цієї суми.

### **Analogy-based estimations**

Оцінка проекту з допомогою естімейтів схожих проектів. Одні типи робіт порівнюються по об’ємах в відсотковому відношенні.

### **Story based estimations**

В основі лежить порівняння сторь в сторі поінтах з використанням Agileрозробки. Найпростіша задача оцінюється в найменшу кількість поінтів, решту оцінюються в порівнянні з використанням чисел Фібоначчі.

### **Cone of Uncertainty (Конус Невпевненості)**

(chapter 4)

Це конусоподібний графік (оординат: множник переестімейту та недоестімейту; абсцис: стадія розвитку проекту), який каже що реальні естімейти з’являються ближче до кінця проекту (кеп☺). Насправді воно так, але звузити той конус можна, якщо всі дизайн активності, проводити на початку проетку, та естімейтити виходячи з точних рекваєментів.

### **Source of Estimation ErrorsCount, Compute, Judge techniques**

* Неточна інформація в рекваєментах;
* Неточна інфа про можливості компанії, яка реалізовує продук;
* Постійні зміни в рекваєментах;
* Неточності сасого процесу естімейту.

### **Diseconomies of Scale**

Основна задача: зрозуміти що оцінювати (величина повинна тісно корелювати з розміром проекту) і порівнювати: в одному середовищі це кількість стрічок коду, десь – кількість веб-сторінок.

### **Delphi method**

* + - 1. Кілька експертів окремо естімейтять проект;
      2. Експерти зустрічаються і обговорюють оцінки. Одного з них вибирають адвокатом;
      3. Оцінки наново анонімно збираються і представляються;
      4. Потім ці оцінки знову обговорюються і відбувається анонімне головування середньої оцінки;
      5. Якщо хоча б одна оцінка сильно відрізняється, то знову повертаються до пункту 2.

### **Challenges with Estimating Size, challenges with Estimating Effort, challenges with Estimating Schedule**

* Спроба оцінити щось без розуміння що це;
* Слухати оцінки тих, хто найбльше кричить;
* Оцінювати повинні програмісти, які це роблять, а не хтось інший;
* Потрібно враховувати ризики, такі як хвороби, мітинги, відпустки, зміни команди;
* Вибір правитльного об’єкту оцінювання;
* Оцінювання є суб’єктивним.

### **Story based scope definition: scoping project, release planning**

Спочатку визначається скоуп всього проекту, а потім відбувається детальніше планування на реліз пленінгу.

### **Documenting and presenting estimation results**

* Представлення в випадку діапазону;
* Оцінка в грубих показниках (місяці, роки, тижні замість днів і годин);
* Оцінювання негативного, позитивного і поточного сценарію;

### **PERT analysis**

**Program (Project) Evaluation and Review Technique** (скорочено — **PERT**) — техніка оцінки та аналізу [програм](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0) ([проектів](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82_%28%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%96%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B0%D0%BC%D0%B8%29)), яка використовується при [управлінні проектами](https://uk.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%96%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B0%D0%BC%D0%B8). PERT — це спосіб аналізу завдань, необхідних для виконання [проекту](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82_%28%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%96%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B0%D0%BC%D0%B8%29). Особливо, аналізу часу, який потрібен для виконання кожної окремої задачі, а також визначення мінімального необхідного часу для виконання всього проекту.

PERT був розроблений головним чином для спрощення планування на папері та ствоерння графіків великих і складних проектів.PERT призначений для масштабних, унікальних, складних, нерутинних проектів.Метод передбачав наявність невизначеності, даючи можливість розробити робочий графік проекту без точного знання деталей і необхідного часу для всіх його складових.

Найпопулярнішою частиною PERT є [Метод критичного шляху](https://uk.wikipedia.org/w/index.php?title=%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BA%D1%80%D0%B8%D1%82%D0%B8%D1%87%D0%BD%D0%BE%D0%B3%D0%BE_%D1%88%D0%BB%D1%8F%D1%85%D1%83&action=edit&redlink=1), що спирається на побудову [мережного графіку](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%80%D0%B5%D0%B6%D0%B5%D0%B2%D0%B8%D0%B9_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%BA) (мережеві діаграми PERT).

# **Solution Design**

## OOD for Test Automation

### **Abstraction**

**Абстракція** – виділяє істотні характеристики певного об’єкта, відрізняючи його від всіх інших видів об’єктів, чітко описуючи його концептуальні границі з точки зору наглядача.

**Модульність –** це розідлення програми на фграгменти, які компілюються окремо, але зв’язані між собою. В ООП модульність має інший характер, оскільки вона передбачає фізичну упаковку класів і об’єктів, тобто – це властивість системи, розкладеної на цілі, але слабо зв’язані між собою модулі.

**Агрегація** створює міцний інструмент який дозволяє фізично об’єднати логічно зв’язані структури а **Наслідування** – дозволяє повторно використовувати ці загальні групи в різних абстракціях.

**Об’єкт –** має стан, демонструє визначену поведінку і володіє унікальними особливостями.

**Стан об’єкта –**характеризує всі (як правило статичних) властивостів об’єкта і поточних (як правило динамічних) значеннями кожної із цих властивостей.

**Поведінка –** це дія і реакція об’єкта, зумовленні через зміну стану об’єкта і передачу помідомлень.

**Клас –** це багато об’єктів, які володіють загальоною структурою, поведінкою і семантикою.

**Поліморфізм –** це концепція теорії типів, відповідно до якого, одне і те саме ім’я може означати екземпляри різних класів зв’язаниз із загальним суперкласам.

**Separation of concerns principle -** один з основоположних принципів інженерної справи. Зокрема - програмної інженерії.

В інформатиці поділ відповідальностей є процес поділу комп'ютерної програми на функціональні блоки, як можна менше перекривають функції один одного. У більш загальному випадку, поділ відповідальностей - це спрощення єдиного процесу рішення задачі шляхом поділу на взаємодіючі процеси за рішенням підзадач.

Поділ відповідальностей дозволяє знизити системну складність, підвищити надійність і адаптивність (гнучкість) програм, забезпечити можливість їх повторного використання.

Будь-яка парадигма програмування полягає в поділі відповідальностей. У разі процедурного програмування - між процедурами і функціями, ООП - між об'єктами, Аспектно-орієнтоване програмування додає до цього списку аспекти.

Шаблони проектування, такі як MVC, забезпечують Поділ подання та змісту.

Сервіс-орієнтована архітектура (SOA) передбачає поділ відповідальності між сервісами

Абстракція - концепт чи ідея, який не є асоційований з певним конкретним об\*'єктом.

Абстрактний клас в C# означає що клас повинен бути унаслідуваний. Абстрактні класи найкраще використовувати коли більша частина класу є загальною для всіх нащадків, а решта має специфічну реалізацію для конкретних об'єктів. Також абстракція допомагає усунути залежності між класами. Тобто коли відбуваютсья зміни в одному класі, то вони не зачіпають ті зміни в іншому класі.

Якщо хоч один метод в класі є абстрактним, то він повинен бути оголошеним як абстрактний клас. Абстрактні методи не містять реалізації.

Якщо абстрактний клас наслідує віртуальні методи з базового класу, то віртуальні методи можуть бути оверрайднуті абстрактними методами.

Абстрактний метод є автоматично віртуальним і при наслідуванні абстарктний клас від класів-нащадків вимагає оверрайду для кожного класу.

**Single responsibility principle -** ажливий принцип [об'єктно-орієнтованого програмування](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), який означає, що [клас](https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) має бути створений для виконання лише однієї задачі, яку він повинен повністю [інкапсулювати](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D1%96%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)). Отже, всі сервіси цього класу мають бути повністю підпорядковані її виконанню. Результатом слідування цій концепції є наявність лише однієї причини для зміни класу, що робить його значно «здоровішим».

Серед плюсів варто відмітити наступні:

* протидіє дублюванню коду, адже якщо функціональність розташована в неправильному місці, то доведеться копіювати її в потрібне;
* зменшує можливість зміни вже відтестованого коду;
* забезпечує відповідність назв класів та їх функціональності, що полегшує життя тим, хто обслуговуватиме цей код в майбутньому.

Мінус використання полягає в зростанні кількості класів, що приводить до зростання складності системи.

**Принцип інве́рсії зале́жностей** ([англ.](http://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0)*DependencyInversionPrinciple*) — один з п'яти [SOLID](http://uk.wikipedia.org/wiki/SOLID_(%D0%BE%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F))-принципів об'єктно-орієнтованого проектування програм, суть якого полягає у розриві [зв'язності](http://uk.wikipedia.org/wiki/%D0%97%D0%B2%27%D1%8F%D0%B7%D0%BD%D1%96%D1%81%D1%82%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) між програмними модулями вищого та нижчого рівнів за допомогою спільних [абстракцій](http://uk.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%B3%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)).

**SOLID** — це п'ять основоположних принципів проектування. Літери слова S.O.L.I.D дуже часто розділяють крапками, тому що насправді - це [абревіатура](http://uk.wikipedia.org/wiki/%D0%90%D0%B1%D1%80%D0%B5%D0%B2%D1%96%D0%B0%D1%82%D1%83%D1%80%D0%B0), складена з перших літер цих п'яти принципів:

* SRP - принцип єдиного обов'язку. Кожен об'єкт має виконувати лише один обов'язок.
* OCP - принцип відкритості\закритості. Програмні сутності повинні бути відкритими для розширення, але закритими для змін.
* LSP - об'єкти в програмі можуть бути замінені їх нащадками без зміни коду програми.
* ISP - принцип розподілення інтерфейсу - багато спеціалізованих інтерфейсів краще за один універсальний. Клієнт, який реалізовує інтерфейс не повинен бути обтяженим зайвими методами, які йому не потрібні. Саме через гранулярність інтерфейсів можна програмувати на різних рівнях.

Наприклад:

1) якщо потрібно проходитись по колекції, використовувати LINQ - юзаємо IEnumerable<T>;

2) якщо додавати, видаляти елементи з колекції, рахувати кількість - ICollection <T>;

3) сортувати, отримувати елемент по індексу - IList<T>;

* DIP - принцип інверсії залежностей. Залежності всередині системи будуються на основі абстракцій, що не повинні залежати від деталей, навпаки, деталі мають залежати від абстракцій. Модулі вищих рівнів не залежать від модулів нижчих рівнів.

### **Encapsulation**

Інкапсуляція – механізм програмування, який ховає деталі реалізації від коричтувача об'єкту. Засоби реалізації інкапсуляції:

* модифікаторидоступу (public, private, internal, protexted);
* проперті(get(аксесор) , set(мутатор));

### **Inheritancevs. Aggregationvs. Composition**

**Наслідування**(механізм "ISA" relationship) – тип відношення між класами, при якому члени базового класу можуть використовуватись класом-нащадком. По-суті це є розширення функціональності базового класу з допомогою класів-нащадків.

Асоціація (механізм "HAS A" relationship)

* **Композиція** (механізм "HAS A"(а точніше "Part Of") relationship) - тип відношення між класами, при якому ціле явно контролює час життя частини. Іншими словами це є часткова реалізація наслідування. Наприклад є класи Engine&Car. Engine є частиною Car. Якщо Car є знищений, то Engine знищується автомвтично. Тобто при цьому типу зв'язку ціле повністю контролює період існування об'єкту частини.

public classEngine  
{  
 . . .   
}

publicclassCar

{

Engine e = newEngine();

.......

}

Переваги композиції:

1. Нема конфлікту імен, який є можливим при наслідуванні;
2. Можливість зміни агрегованого об'єкту в runtime;
3. Повна заміна агрегованого об'єкту в класах, які є нащадками від класу, що вміщає агрегований об'єкт;
4. Є можливість використання функціоналу інших класів через створення об'єктів всередині класу;
5. Є можливість скрити певну частину реалізації, а також вихідні параметри, необхідні поведінці через передачу

* **Агрегація** (механізм "HASA" relationship) - тип відношення між класами, при якому зв'язок між цілим і частиною є доволі слабким, оскільки час життя об'єктів цілого і частини не є пов\*язані (наприклад скалдова частина передаєтсья через параметр конструктора). Наприклад є 2 ентіті: Customers і Address. В цьому випадку Customers "has an" Address. Час життя частини (Address) не контролюється цілим (Customers).

public classAddress  
{  
 . . .  
}

publicclassPerson

{

private Address address;

public Person(Address address)

{

this.address = address;

}

. . .

}

Ціоб'єктизгодомбудутьстворюватисьокремо:

Address address = new Address();  
Person person = new Person(address);

### **Modularity**

Модульність - принцип, згідно якого програма ділиться на окремі іменовані сутності. Він часто використовується для спрощення задач програмування і проектування між групами програмістів. Код прогрми розбивається на декілька файлів, кожен з яких компілюється окремо від інших. Така модульність дозволяє значно зменшити час перекомпіляції при внесенні змін в окремі файли і спрощує групову розробку. Також цей принцип дозволяє замінювати окремі компоненти кінцевого програмного коду без необхідності перезбірки всього проекту.

### **Separation of concerns**

Це принцип програмної архітекутри розбиття комп'ютерної програми на окремі модулі, при якому окремі компоненти мінімально перетинаються та залежать один від одного, а також дублюють функціонал один одного. При цьому кожен модуль відповідає за роботу в конкретних визначених рамках, тобто відповідати за конкретну задачу. В основі цього принципу лежить часте використання інтерфейсів. Хорошим прикладом є MVC дизайн паттерн. Переваги:

- дозволяє працювати з частиною функціоналу ізольовано;

- легко вносити зміни та підтримувати код;

Види відповідаьностей:

* Core concerns - бізнес-логіка аплікації, випливає безпосередньо з функціональних вимог.
* Cross-cutting concerns - функціональність, яка не відноситься до бізнес-логіки. Випливає восновному з нефункціональних вимог, а також з внутрішніх потреб (розширення, підтримка). Тісно переплітаєтсья з основною функціональністю.

### **Types vs. Classes**

An object’s class defines how the object is implemented .The class defines object's internal state and the implementation of its operations. In contrast, an object’s type only refers to its interface -the set of requests to which it can respond. An object can have many types, and object of different classes can have the same type.

Тип є абстрактним інтерфейсом, ві не містить реалізації.

Клас є реалізацією типу. Він визначає поведінку об’єкта.

Об’єкти різних класів можуть мати один тип якщо вони, наприклад, реалізують один і той же інтерфейс.

### **Functional logic organization patterns**

**Repository pattern**

It’s a layer that allows you to separate application from data storage. Other words: your code doesn’t work directly with data storage.

**UI Automation Page Object Pattern**

Page Object pattern - шаблон проектування, при якому об'єкти класів моделюють сторінки аплікації і поведінку юзера. Логіка тестів існує окремо від опису сторінки та реалізації дій користувача.Це забезпечує легку підтримку коду та допомагає уникнути дублювання коду.

* Об'єкти сторінок не повинні здійснювати перевірки, ця частина повинна знаходитись завжди в тесті.
* Єдина перевірка, яка повинна включатися в об'єкт сторінки, це перевірка на коректну загрузку сторінки. Цю перевірку потрібно розміщати в конструкторі об'єкту сторінки.

**Composite Page Object**

Це паттерн, який базується на використанні об'єктів певних компонент і інших класах. Наприклад створення екземплярів класу NavigationPanel в класах HomePage та CreatePatientPage, де NavigationPanel ввикористовується.

**ExtendedPageObject**

Це паттерн, який базується на наслідуванні базової сторінки сторінками-нащадками і є альтернативою до CompositePageObject.

**PageObject в Selenium**

Цей паттерн реалізований в селеніумі з допомогою бібліотеки PageFactory і класу сторінки. Пошук елементів на сторінці можна здійснювати з допомогою методу <findElement>, а можна з допомогою директиви <@FindBy>, яка працює тільки з використанням PageFactory. Ініціалізація елементів відбувається при зверненні до нього на сторінці. Для цього PageFactory використовує LAZY ініціалізацію.

**UI Maps**

UIMap - зв, який містить всю інформацію про UI контроли аплікації, що тестується. Використовуються для автоматизованого тестування UI десктопних аплікацій.

**SUT Runner**

Це об'єкт, який відповідає за запуск та закриття тестованої системи. Дозволяє початкову ініціалізаію об'єкту системи (наприклад відкриття браузера та навігацію на стартову сторінку) та вивільнення ресурсів після закінчення тестів (закриття браузера.)

**DDT**

Це вид тестування, при якому один і той же тест виконується багато разів з різними наборами даних. Набори даних, як правило, зберігаються в зовнішніх файлах.

### **Automation Testing Pyramid**

Суть цієї піраміди полягає в тому, що потрібно мати багато низькорівневих (unit+integration) тестів, ніж end-to-end тестів, які проганяються через UI.

* Unittesting - це тести, які перевіряють роботу якоїсь конкретної функції. Виконується переважно для перевірки логіки роботи конкретної функції після змін коду та визначити чи відповідає конкретна функція бізнес рекваєментам та архітектурі. Юніт тести допомагають знаходити помилки на початковому етапі розробки. Переважно імплементуються девелоперами. Вони швидкі і відносно дешеві в реалізації.
* Integrationtesting - тести, які перевіряють спільну роботу декількох функцій. Виконується переважно для перевірки роботи групи функцій, які реалізують певний функціонал. Виконуєтсья після модульного тестування.
* Acceptance testing - це black-box тести. Вони перевіряють роботу аплікації суто з точки зору бізнесу. Для програм з GUI це перевірка фловів поведінки користувача на GUI. Вони найбільш зрозумілі стейкхолдерам, проте ламаються при зміні GUI і є найдорожчими в реалізації і підтримці.

Рекомендовано більше код покривати Unit-тестами (наприклад 70%), потім integration (20%), а потім - acceptance (10%). Більшість функціоналу пропонується покривати юніт тестами, бо вони найшвидші і їх найлегше підтримувати. Аксептанс тести ранятсья довше і їх тяжче підтримувати. Тому їх кількість повинна зводитись до мінімуму. Аксептанс тестами варто покривати end-to-end сценарії.

В тестуванні дуже важливо зрозуміти що є важливим і протестувати це.

### **GoF Design Patterns**

Pattern – це шаблон вирішення якоїсь конкретної проблеми таким чином, що цим шаблоном можна користуватись длявирішення. Вони поділяються на:

* Creational (Singleton, Factory Method, Builder);
* Structural (Decorator, Adapter, Facade);
* Behavioral ().

Породжуючий патерн.

**Синглтон** забезпечує існування єдиного екземпляру класу та єдиного доступу до нього. (Гарантує, що в класу є тільки один екземпляр, та надає до нього глобальну точку доступу.)Уявімо, що нам потрібна глобальна логувальна система в програмі, також нам треба логувати наші повідомлення в якийсь один-єдиний файл, причому нумеруючи повідомлення послідовно, незалежно від того, з якого закутка програми вилетіла помилка. В основі патерну – статичний об’єкт певного типу, доступ до якого надається через статичний метод класу.

class LoggerSingleton {

**private LoggerSingleton() { }**

int logCount = 0;

**private static LoggerSingleton \_loggerSingletonInstance = null;**

**public static LoggerSingleton GetInstance() {**

if (\_loggerSingletonInstance == null) {

\_loggerSingletonInstance = new LoggerSingleton(); }

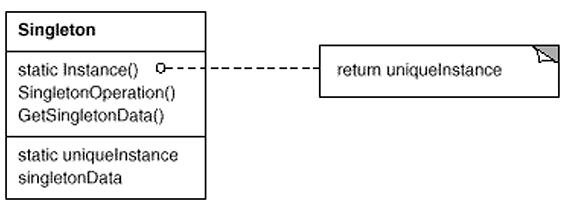
return \_loggerSingletonInstance; }

public void log(String message) {

System.out.println(message);

\_logCount++; } }

Виклик LoggerSingleton logger = LoggerSingleton.GetInstance();



Використовувати паттерн, коли :

* Повинен існувати тільки один екземпляр деякого класу легко доступний всім клієнтам;
* Єдиний екземпляр повинен розширюватися шляхом породження підкласів, і клієнтам потрібно мати можливість працювати з розширеним екземпляром без модифікації свого коду.

Переваги:

* Дозволяє контролювати доступ до єдиного екземпляру
* забезпечує зменшення кількості імен(дозволяє уникнути засмічення простору глобальних змінних, в якх зберігаються унікальні екземпляри)
* допускає змінне число екземплярів класу

Недоліки:

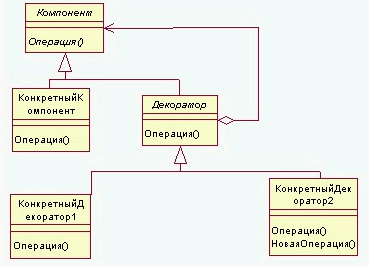
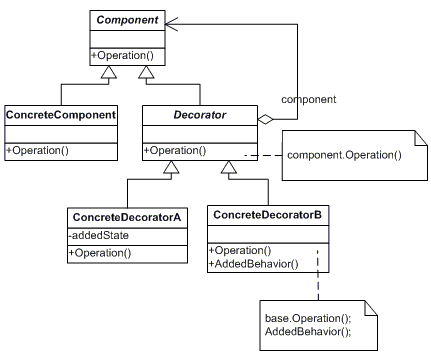
* глобальні об’єкти можуть бути шкідливими для об’єктного програмування
* ускладнює написання модульних тестів та TDD

**Декоратор**

Структурний патерн. Динамічно додає об'єкту нові обов'язки. Є гнучкою альтернативою породженню підкласів з метою розширення функціональності.

**Використовуйте патерн декоратор:**

* для динамічного, прозорого для клієнтів додавання обов'язків об'єктам
* для реалізації обов'язків, які можуть бути зняті з об'єкта;
* коли розширення шляхом породження підкласів з якихось причин незручно або неможливо.

### «Компонент» визначає інтерфейс для об’єктів, на які можуть бути динамічно покладені додаткові обов’язки,

### «КонкретнийКомпонент» визначає об’єкт, на який покладаються додаткові обов’язки, «Декоратор» – зберігає посилання на об’єкт «Компонент» і визначає інтерфейс, відповідний інтерфейсу «Компонента».

### «КонкретнийДекоратор» покладає додаткові обов’язки на компонент.

### «Декоратор» переадресує запити об’єкту «Компонент».

### Переваги:

### Більша гнучкість, ніж у статичного спадкування: можна додавати і видаляти обов’язки під час виконання програми в той час як при використанні наслідування треба було б створювати новий клас для кожного додаткового обов’язку.

### Даний патерн дозволяє уникнути перевантажених методами класів на верхніх рівнях ієрархії – нові обов’язки можна додавати в міру необхідності.

### Недоліки:

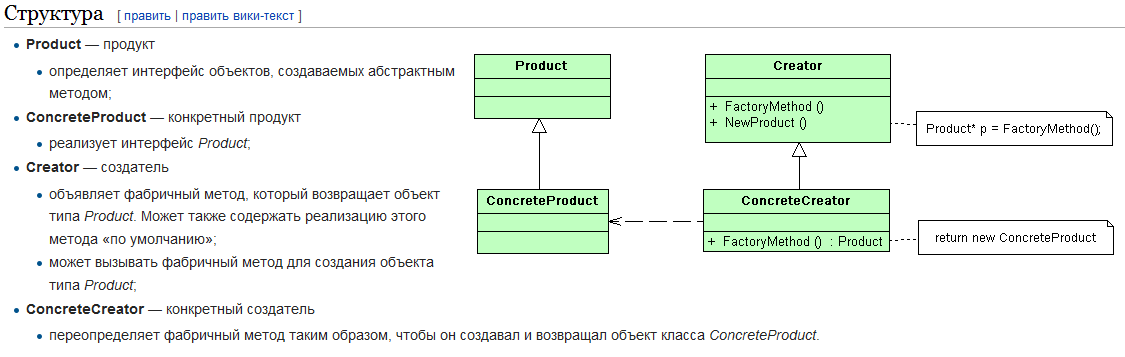
### «Декоратор» і його «Компонент» не ідентичні

система складається з великої кількості дрібних об’єктів, які схожі один на одного і розрізняються тільки способом взаємозв’язку а не класом і не значеннями своїх внутрішніх змінних – така система складна в вивчення та налагодженні

**Фабричний метод**

Породжуючий паттерн.. Визначає інтерфейс для створення об'єкта, але залишає підкласами рішення про те, який клас ініціалізувати(приймаючи якийсь параметр і вони просто повертають екземпляр відповідного конкретного продукту).

**Використовуйте патерн фабричний метод, коли:**  
А класу заздалегідь невідомо, об'єкти яких класів йому потрібно створювати;  
а клас спроектований так, щоб об'єкти, які він створює, специфіковані підкласами;  
а клас делегує свої обов'язки одному з декількох допоміжних підкласів, і ви плануєте локалізувати знання про те, який клас приймає ці обов'язки на себе.



**Переваги:**

* фабричні методи позбавляють проектувальника від необхідності вбудовувати  
  в код класи які залежать від програми. Код має справу тільки з інтерфейсом  
  класу Product, тому він може працювати з будь-якими певними користувачами класами конкретних продуктів.
* дозволяє зробити код створення об'єктів більш універсальним, не прив'язуючись до конкретних класах (ConcreteProduct), а оперуючи лише загальним інтерфейсом (Product);

**Недолік:**

* що клієнтам, можливо, доведеться створювати підклас класу Creator для створення лише одного об'єкта ConcreteProduct (потрібно створювати новий підклас лише для того, щоб змінити клас продукту.)
* **Фасад**

Фасад – це патерн, який структурує об’єкти, надає уніфікований інтерфейс замість наборів інтерфейсів деякої підсистеми. Фасад визначає інтерфейс більш високого рівня, який спрощує використання підсистеми. (**Фасад** надає єдину «точку доступу» до підсистеми, тим самим спрощуючи її використання та розуміння.)



Клієнти спілкуються з підсистемою, посилаючи запити фасаду. Він переадресує їх відповідним об'єктам усередині підсистеми

**Використовуйте патерн фасад, коли:**

* хочете надати простий інтерфейс до складної підсистемі.;
* між клієнтами і класами реалізації абстракції існує багато залежностей.
* ви хочете розкласти підсистему на окремі шари. Використовуйте фасад для визначення точки входу на кожен рівень підсистеми.

**Переваги:**

* Ізолює клієнтів від компонентів системи
* дозволяє послабити зв'язаність між підсистемою і її клієнтами. Слабка зв'язаність дозволяє видозмінювати компоненти, не зачіпаючи при цьому клієнтів.
* фасад не перешкоджає додаткам безпосередньо звертатися до класів підсистеми, якщо це необхідно. Таким чином, у вас є вибір між простотою і спільністю.

**Недоліки**

* ??Хоча основну роботу виконують саме об'єкти підсистеми, фасаду, можливо, доведеться перетворити свій інтерфейс в інтерфейси підсистеми.
* Не підходить якщо необхідно забезпечити скритність і акуратність (прозорість) вибором по відношенню до іншої групи об'єктів ( більш підходящим є шаблон Проксі).

**Будівельник**

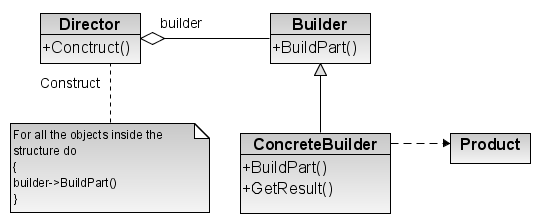
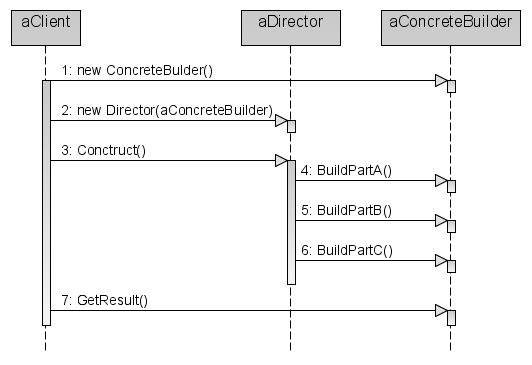
## *Породжуючий патерн. Відокремлює конструювання складного об'єкта від його подання, таким чином у результаті одного й того ж процесу конструювання можуть бути отримані різні подання.*

Вам слід створити систему, що дозволить легко *будувати* будь-яку конфігурацію ноутбука для будь-якого покупця. Причому стандартні конфігурації мають складатися «по накату»( для геймера або для подорожей). Звісно, що ви наперед знаєте певні кроки, щоб створити ноутбук. Ці кроки визначаються в *абстрактному будівельнику* (*abstract builder*). Єдине що вам потрібно від клієнта, це дізнатися, які компоненти використовувати на кожному кроці.

## Слід використовувати шаблон Будівельник коли:

### **алгоритм створення складного об'єкта не повинен залежати від того, з яких частин складається об'єкт та як вони стикуються поміж собою;**

процес конструювання повинен забезпечити різні подання об'єкта, що конструюється

### **Builder — будівельник: визначає абстрактний інтерфейс для створення частин об'єкта Product;**

### **ConcreteBuilder — конкретний будівельник: конструює та збирає докупи частини продукту шляхом реалізації інтерфейсу Builde(будує внутрішнє подання продукту та визначає процес його зборки;)**

### **Director — управитель: конструює об'єкт, користуючись інтерфейсом Builder; Product — продукт: подає складний конструйований об'єкт.**

**Переваги:**

- Дозволяє змінювати внутрішнє представлення продукту; (Оскільки продукт конструюється через абстрактний інтерфейс, то для зміни внутрішнього представлення достатньо всього лише визначити новий вид будівельника;)  
- ізолює код, який реалізує конструювання та представлення;  
- Дає більш тонкий контроль над процесом конструювання (На відміну від породжують патернів, які відразу конструюють весь об'єкт цілком, будівельник робить це крок за кроком під управлінням уравителя. І лише коли продукт завершений, розпорядник забирає його у будівельника.).

**Мінуси:**

ConcreteBuilder і створюваний ним продукт жорстко пов'язані між собою, тому при внесенні змін в клас продукту швидше за все доведеться відповідним чином змінювати і клас ConcreteBuilder.

### **Keyword-driven testing**

Це методологія, яка використовує ключові слова для відображення того що тестується. Це візуальне представлення тестових скриптів, при якому кожній діє відповідає ключове слово. Декілька ключових слів об’єднуються в дії. Наприклад EnterClient - це набір кроків, які повинні бути виконані для інсерту нового клієнта в базу даних.

Точно так як в DDTми відділяємо дані від коду так само в KDTми відділяємо тести від коду. Це робиться для того щоб нетехнічні люди могли писати авто тести без особливих навичок та створення тестів, які не залежать від конкретного інструменту.

Тулзи для Keyword-driventesting: TestComplete, SeleniumIDE, SilkTest.

### **Anti-Patterns**

Це не ефективний спосіб вирішення поставленої задачі. Найважливіша мета документації антипаттернів — полегшити можливість розпізнавання шкідливого рішення і успішного виправлення помилок ще на ранніх етапах роботи

**Spaghetti code** - слабо структукрурована і погано спроектована система, яка тяжка для розуміння. Фікс: розділяти функціонал по різних класах і файлах щоб він був читабельним і зрозумілим.

**God object** - створення об'єкту, який бере на себе дуже багато функцій і зберігає в собі основну кількість даних. Фікс: розбивати задачі на підзадачі з можливістю вирішення їх різними розробниками.

**Big ball of mud** - ситуація, яка трапляєтсья переважно через зміну рекваєментів, девелоперів чи їх некомпетентність. Це випадок, коли ти все залежить від всього і при внесенні змін в код тяжко спочатку чітко визначити де потрібно вносити зміни. Часто так відбувається коли через погано задокументовані рекваєменти вибирається невідповідна архітектура солюшина. Фікс: вимагати чіктих рекваєментів на стадії дизайну системи.

**Magic strings -** використання захардкоджених строкових значень в коді без чіткого коментаря для чого це зроблено. Людині, яка не девелопила цей код, потім тяжко поянити звідки воно взялось. Буває інколи що при зміні цієї стрічки код престає працювати взагалі чи викликає якусь приховану функцію яка спрацьовує при введені якихось специфічних значень. Фікс: не використовувати константи в коді без належних для того причин і відповідних коментарів.

**Magic numbers** - використання захардкоджених числових констант в коді без чіткого коментаря для чого це зроблено. Людині, яка не девелопила цей код, потім тяжко поянити звідки воно взялось. Буває інколи що при зміні ціього числа код престає працювати взагалі. Фікс: не використовувати константи в коді без належних для того причин і відповідних коментарів.

**Copy paste programming** - Дублювання коду при створенні двох схожих функцій. Потім потрібно вносити зміни в кількох місцях, кількість коду також необгрунтовано збільшується.

### **Inversion Of Control**

**Inversion of Control (инверсия управления)** — это некий абстрактный принцип, набор рекомендаций для написания слабо связанного кода. Суть которого в том, что каждый компонент системы должен быть как можно более изолированным от других, не полагаясь в своей работе на детали конкретной реализации других компонентов.

**Dependency Injection (внедрение зависимостей)** — это одна из реализаций этого принципа (помимо этого есть еще [Factory Method](http://en.wikipedia.org/wiki/Factory_pattern), [Service Locator](http://en.wikipedia.org/wiki/Service_locator_pattern)).

Інверсія управління - принцип організації програми, згідно з яким керування потоку виконання програми здійснюється окремим модулем, який дозволяє зменшити зв'язаність між іншими модулями програми. Тобто кожен компонент системи повинен бути максимально ізольованим та не залежати від деталей реалізації інших компонентів.

### **Dependency Injection**

Design Patterns

* Constructor Injection
* Property Injection
* Method Injection
* Service Locator

Dependency Injection Containers

Unity, StructureMap, Autofac, Ninject, Castle Windsor...

Найвідомішою реалізацією Ioc є Dependency Injection principle , який використовується в багатьох фреймворках, які називаютсья Ioc контейнери (є ще Factory Method, Service Locator). Суть полягає в тому, що модулі верхніх рівнів не повинні залежати від модулів нижчих рівнів. Обидва типи модулів повинні залежати від абстракцій. Абстиракції не повинні залежати від деталей. Деталі повинні залежати від абстаркцій.

IoC-контейнер - бібліотека, яка дозволяє максимально спростити та автоматизувати процес написання коду. З їх допомогою задається відповідність між інтерфейсом і його конкретною реалізацією, щоб потім всюди нею користуватись.

Згідно практики, клієнт, який використовує певний сервіс, повинен це робити ненапряму, а через посередника. У випадку змін коду нам достатньо буде змінити сам сервіс, але не протрібно буде нічого міняти в клієнті.

interface IScheduleManager { Schedule GetSchedule(); }

class ScheduleManager : IScheduleManager

{ public Schedule GetSchedule() { // Do Something by init schedule... } }

class SimpleConfigModule : NinjectModule

{ public override void Load()

{ Bind<IScheduleManager>().To<ScheduleManager>();}}

class ScheduleViewer

{

}

Main()

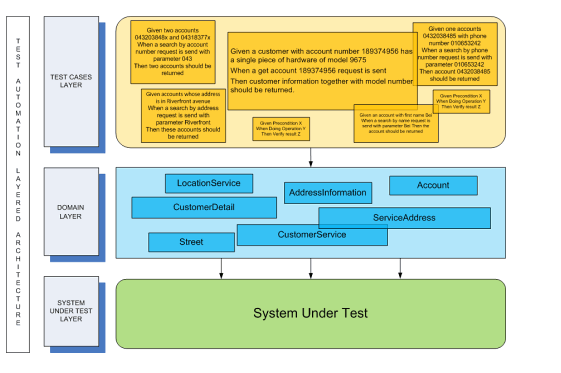
{

IKernel ninjectKernel = new StandardKernel(new SimpleConfigModule());

ScheduleViewer scheduleViewer= ninjectKernel.Get<ScheduleViewer>();

**Layered Test Solution (Abstraction layers)**

The test automation code must:

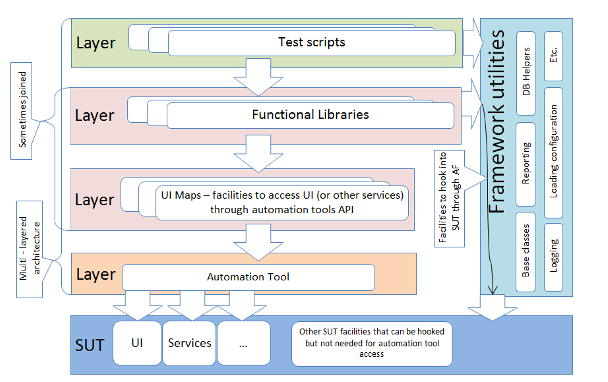
1. Assemble a URL based on the operation under test,
2. Send out a http request with some http libraries,
3. Interpret the response sent back from the web server and parse the xml,
4. Compare results returned to expected results.

### **Multi Layered Test Solution**

Вона розділяє логіку системи тестування в окремі логічні рівні.

Це широко поширена практика для поділу програмної системи архітектурно на окремі шари. Перший рівень інкапсулює логіку викладу, другий рівень бізнес-логіки рівень і третій рівень відповідає за зберігання даних. Використання цієї парадигми дозволяє знизити вартість обслуговування додатків, так як компоненти всередині кожного рівня можуть бути змінені з мінімальним впливом на інших рівнях. Такий же підхід може бути застосований до системи тестування.

Тестовий код може бути розділений на три шари: шар інтерфейсу інструменту автоматизації призначеного для користувача інтерфейсу для тестованої системи (SUT) доступу, шар функціональної логіки і тестовий приклад шар. Кожен шар має певну відповідальність із загальною метою зменшення витрат на утримання тестів і полегшення створення нового тесту.



**Fluent interface**

Cпосіб реалізації об'єктно-орієнтованого API, націлений на підвищення читабельності вихідного коду програми. Назва придумано Еріком Евансом і Мартіном Фаулером.

Поточний інтерфейс хороший тим, що спрощується множинний виклик методів одного об'єкта. Зазвичай це реалізується використанням ланцюжка методів, що передають контекст виклику наступної ланки (але текучий інтерфейс тягне за собою щось більше, ніж просто ланцюжок методів). Зазвичай, цей контекст:

- визначений за допомогою значення, що повертається методом;

- успадковується (як нового контексту використовується попередній);

- припиняється поверненням нічого не значущого значення (void).

Такий стиль побічно корисний підвищенням наочності і інтуїтивності кодую Однак може дуже згубно позначитися на налагодженні, якщо ланцюжок діє як один вислів, куди відладчик не завжди може встановити проміжну точку зупину.

# Automation Development\Automation concept

## Automation concept

### AutomationDefinition

Це тестування програмного забезпечення з використанням написаного тестового коду на певній мові програмування чи автоматизованих тулзів.

### AutomationGoalsandTasks

Цілі повинні бути вимірювані, тому кожна ціль повинна мати якесь кількісне відображення. Основними цілями повинно бути зменшення кількості багів на продакшині, зменшення часу та витрачених грошей при регресійному тестуванні. А саме:

* Максимальне зменшення кількості мануальних кейсів для прискореного проведення regression чи verification тестування, особливо при тестуванні на різних середовищах;
* Збільшення покриття тестових випадків;
* Покращує розуміння роботи системи яка тестується, що призводить до зменшення кількості багів на продакшині;
* Знаходження багів швидше і раніше якщо тест раниться на дев машині при поставленому CIпроцесі що призводить до економії часу і грошей порівняно з пізнім знаходженням бага;
* Підвищення рівня якості через автоматизацію тестування рутинних кейсів, баги в яких можна пропустити через втрату концентрації що призводить до зменшення багів на продакшині та економії часу та грошей. Тести не втомлюються);
* Зменшення кількості дублювання роботи використовуючи key-drivenтестування з використанням тих самих екшинів на різних тулзах що призводить до економії часу та коштів.

Які кейси слід автоматизовувати:

* Кейси з високим пріоритетом;
* Часто повторювані і рутинні;
* Кейси, швидкість перевірки яких з допомогою автоматизації є відчутно швидшою ніж мануально;
* Перевірка одних і тих же сценаріїв на різних середовищах;
* Тестування перфомансу;
* Лоад тестування;

Не доцільно автоматизовувати кейси:

1. Якщо тест потрібно прогнати 1 раз;
2. Якщо код постійно міняється через нечіткі рекваєменти;
3. Якщо під час кейсу потрібне втручання ззовні, наприклад заміна апратури.

### AutomatingPerformance

Тестування перфомансу можна розділити на 2 види:

1. Час на відсилання реквесту та отримання респонсу від сервера (JMeter, LoadRunner);

Принцип роботи Jmeter:

1. Додавання Thread Group:
   1. Створюється Numberofthreads – кількість юзерів(средів) , які будуть працювати з сайтом;
   2. Задається Rump-Up період – період за який повинна стартувати задефайнена кількість средів;
   3. Якщо ThreadUser = 100 і Rump-Up = 100, то затримка буде 1 секунда (100/100 = 1 user/sec).
2. Додавання ConfigElements, задаються семплери відповідно до типу реквесту, урла, IPсервера…
3. Налаштування відображення результатів: оптимально брати AgreagteReport, бо він інформативний і не споживає багато ресурсів машини, що може викривлювати результати.
4. Час на завантаження сторінки (HTTPWatch, BrowserDevTools) – тут можна використовувати замірювання часу вручну + вирахування похибки з HttpWatch. Оскільки після загрузки сторінки ще можуть йти інші Ajaxреквести потрібно визначити що означає сторінка загрузилась – вона видима для юзера чи виконався останній реквест.

При автоматизації сторінок з Ajaxреквестами при загрузці сторінки Seleniumчекає параметр document.Readystate = complete, після якогоще багато може виконуватись на сторінці тому треба писати свої кас томні вейтери для того щоб дочекатися завантаження сторінки. Порядок ініціалізації сторінки є таким:

1. head scrips
2. body scripts
3. onload
4. later things.

Onloadозначає, що такі ресурси як картинки, CSSстилі, скрипти (грубо кажучи - сторінка) були завантажені. Тому тут потрібно визначати похибку мануальними засобами і враховувати її при вимірюванні тулзами.

### UIAutomation

На даний момент існує 2 підходи: запис кейсів з допомого тулзи і написання свого фреймворку, який буде повторювати дії користувача.  
TipsandTricks:

* + - 1. Автоматизація правльних кейсів

Які кейси слід автоматизовувати:

* Кейси з високим пріоритетом;
* Часто повторювані і рутинні;
* Кейси, швидкість перевірки яких з допомогою автоматизації є відчутно швидшою ніж мануально;
* Перевірка одних і тих же сценаріїв на різних середовищах;
* Тестування перфомансу;
* Лоад тестування;

Не доцільно автоматизовувати кейси:

1. Якщо тест потрібно прогнати 1 раз;
2. Якщо код постійно міняється через нечіткі рекваєменти;
3. Якщо під час кейсу потрібне втручання ззовні, наприклад заміна апратури;
4. Кейси з низьким пріоритетом, та протестованийкод , який не буде змінюватись;
   * + 1. Код повинен інкапсульовуватись, а функції в тесті повинні бути читабельними і з них повинні бути зрозумілі дії користувача.
       2. Дуже важливо щоб елементи на сторінці мали постійні локатори, найкраще ID.
       3. Використовувати тест-дизайн техніки для правильного вибору тестових сценаріїв.
       4. Уникати комплексних сценаріїв, які тестуть все, їх потрібно гранулювати.
       5. Тести повинні бути незалежними.
       6. Використовувати спеціальні вейтери для асинхронних операцій.
       7. Намагатись максимально скоротити і спростити степи в тестах виконуванням попередньо прекондишинів і конфігурацій перед тестами.

### Playback/recording instruments for routine tasks

Такі тулзи не є дуже ефективними, оскільки ці кейси доволі тяжко смейнтейнити, вони мають обмеження (клікання по попапах, підтримка браузерів, неможливість використання DDT, умов і килів, неможлива загрузка файлів).Проте можна створити сценарій з допомогою SeleniumIDE, експортнути то в C#(наприклад) файл і потім редагувати вручну.

### Creating a test suite with playback/recording tool

Процедура така ж я к і при мануальному тестуванні, але потрібно пронати і записати кожен кейс.

### Performance VS Load Testing

1. Loadtesting - перевірка стабільності роботи програми при одночасній роботі певної кількості користувачів;
2. Performancetesting - перевірка стабільності роботи програми при виконанні якоїсь кількосіті операцій за одиницю часу;

## Generic Programming Language Knowledge

### Platform execution runtime

**.NetFramework -** програмна технологія, запропонована компанією Майкрософт, для створення і виконання десктопних і веб аплікацій. Одною з ідей .NET є сумісність служб, написаних різними мовами. Кожна бібліотека в .Net має свідчення про свою версію, що дозволяє усунути можливі конфлікти між різними версіями збірок.

Ця платформа складається з двох складових:

1. **CommonLanguageRuntime** – середовище виконання, яке забезпечує виконання збірки. Головна задача CLR полягає в автоматичному визначенні, управлінні і загрузці типів даних Ця віртуальна машина забезпечує виконання програмного коду і пропонує вирішення наступних задач:

* управління кодом (загрузка і виконання);
* управління пам’яттю при розміщенні об’єктів, процесами та потоками;
* ізоляція пам’яті аплікацій;
* перевірка безпеки коду;
* перевірка і динамічна JIT компіляція MSIL коду в команди процесора;
* доступ до метаданих;
* обробка виняткових ситуацій;
* взаємодія між керованим (при виконанні цього коду CLRактивізує управління пам’яттю. Виконується в середовищі CLR) і некерованим (автоматичне управління пам’яттю не відбувається) кодом (в тому числі і COM-об’єктами);
* підтримка сервісів для розробки (debugging).

Після загрузки CLR в адресний простір процесу, ядро середовища виконання виконує наступні дії:

* визначає місцезнаходження збірки;
* загружає збірку в пам’ять;
* проводить аналіз наповнення збірки (виявляє класи, структури, інтерфейси);
* проводить аналіз метаданих;
* забезпечує компіляцію коду з ILв асемблер ний код;
* проводить перевірку безпеки;
* використовуючи основний потік застосунку, передає управління фрагменту коду збірки, перетвореному в команди процесора.

1. **ClassLibrary** – об’єктно-орієнтована бібліотека класів, інтерфейсів і типів за значенням, що відповідає CLS специфікації. Ця бібліотека забезпечує доступ до функціональних можливостей системи і призначена в якості основи при розробці .Netаплікацій, компонент, елементів управління.

### CompiledlanguageswithScriptlanguages

Основна різниця між тими мовами: скриптові(Perl, PHP, Phyton, Ruby) мови не потрібно компілювати, вони інтерпретуються браузером. Вони є крос-платформеними. Скриптовімови не створюють виконавчих файлів (exe-файлів).

Трансляція – переведення високорівневої мови на машинний код. Є 2 види – інтерпретація і компіляція.

Компіляція – процес переведення в рантаймі мови високого рівня на машинний код, виконання якого відбувається після того як програма повністю переведена. Програми на цих мовах є найшвидшими, оскільки після компіляції код виконується на машинній мові.

Інтерпретація – построкове переведення високорівневого коду в машинний код. Програми на таких мовах є повільніші.

### DeclareClasses

Клас – користувацький тип даних, який описує поведінку об’єкта.

Buildinghouse; //створення посилання на об’єкт.

house = newBuiding(); //виділяється пам’ять і створюється об’єкт.

* В с# можна наслідувати лише один клас;
* Назва класу не може починатись з цифри чи спец символа крім “@”;
* Можна наслідувати безліч інтерфейсів;
* Базовим для всіх є клас Object, який провадить базові методи:
  + boolEquals(Objectobj)–поервтає результат порівняння objз поточним об’єктом;
  + GetType() – повертає тип об’єкту в ран таймі (повертає об’єкт типу Type);
  + GetHashCode() – повертає числове значення з Хеш-таблиці, яке використовується для ідентифікації об’єкта при перевірці на рівність. Коли 2 об’єкти мають одинаковий хеш-код, то вони рівні, проте обернена залежність невірна (тобто якщо 2 об’єкти мають одинакові хеш-коди, то не означає що вони рівні).

Хеш таблиця – тип колекції, яка містить пари ключ-значення, а форматі, який забезпечує швидку вставку, пошук і видалення елементів. Одним з прикладів використання є збереження історії в браузерах.

* + ToString() – повертає строкове представлення поточного об’єкта.

### DeclareInterfaces

Інтерфейс – тип даний, який містить тільки сигнатуру методів, пропретей, івентів чи індексаторі. Тобто він описує поведінку об’єкта без конкретної реалізації.

* Він є паблік по-дефолту;
* Об’єкт інтерфейсу можна створити тільки створивши об’єкт класу, який його реалізує;
* Інтерфейс можна наслідувати інші інтерфейси;
* Наслідування багатьох інтерфейсів є дозволеним;
* Є єдиним джерелом наслідування для структур.

### DeclareclassMembers

Членами класу є методи, проперті, події, індексатори, оператори, конструктори, деструктори.

* Конструктори тадеструкториповинні називатись як клас, в якому вони використовуються. Інші члени класу повинні мати унікальні для цього класу імена.

### SimpleVariableTypes

|  |  |  |  |
| --- | --- | --- | --- |
| C# Type | .Net Framework Type | Values | Memory |
| [bool](https://msdn.microsoft.com/en-us/library/c8f5xwh7.aspx) | System.Boolean | True, false |  |
| [byte](https://msdn.microsoft.com/en-us/library/5bdb6693.aspx) | System.Byte | From 0 to 255(2^8-1) | 1 byte |
| [sbyte](https://msdn.microsoft.com/en-us/library/d86he86x.aspx) | System.SByte | From -128 (-2^7) to 127(2^7-1) | 1 byte |
| [char](https://msdn.microsoft.com/en-us/library/x9h8tsay.aspx) | System.Char | - | 2 bytes |
| [decimal](https://msdn.microsoft.com/en-us/library/364x0z75.aspx) | System.Decimal |  | 16 bytes |
| [double](https://msdn.microsoft.com/en-us/library/678hzkk9.aspx) | System.Double |  | 8 bytes |
| [float](https://msdn.microsoft.com/en-us/library/b1e65aza.aspx) | System.Single |  | 4 bytes |
| [int](https://msdn.microsoft.com/en-us/library/5kzh1b5w.aspx) | System.Int32 | From -2 147 483 648 (- 2^31)to 2 147 483 647 (2^31-1) | 4 bytes |
| [uint](https://msdn.microsoft.com/en-us/library/x0sksh43.aspx) | System.UInt32 | From 0 to 4 294 967 295 (2^32 - 1) | 4 bytes |
| [long](https://msdn.microsoft.com/en-us/library/ctetwysk.aspx) | System.Int64 | From -9 223 372 036 854 775 808(-2^63)to 9 223 372 036 854 775 807 (2^63 - 1) | 8 bytes |
| [ulong](https://msdn.microsoft.com/en-us/library/t98873t4.aspx) | System.UInt64 | From 0 to 18 446 744 073 709 551 615 (2^64 - 1) | 8 bytes |
| [object](https://msdn.microsoft.com/en-us/library/9kkx3h3c.aspx) | System.Object | - |  |
| [short](https://msdn.microsoft.com/en-us/library/ybs77ex4.aspx) | System.Int16 | From -32768 (-2^15)to32767(2^15 - 1) | 2 bytes |
| [ushort](https://msdn.microsoft.com/en-us/library/cbf1574z.aspx) | System.UInt16 | From 0 to 65 535 (2^16 - 1) | 2 bytes |
| [string](https://msdn.microsoft.com/en-us/library/362314fe.aspx) | System.String | - | 2 bytes/char |

Різниця між типом за значенням та типом за посиланням: при копіюванні типу за значенням створюється копія значення. При копіюванні типу за значенням створюється нове посилання на ту саму область в пам’яті.

Типи за значенням розташовані в стеку. Типи за посиланням - в купі. При використанні типів за посиланням посилання на об'єкт створюється в стеку, а об'єкт вже створюєтсья в купі.

Для збереженні даних в стеку використовується статична пам'ять (SRAM). В купі - динамічна пам'ять (DRAM). Статична і динамічна пам'ять відносяться до оперативної пам'яті, проте статична пам'ять працює швидше, оскільки дані повністю розміщені в ній.

**Статична** пам'ять (SRAM) - як павило, використовується як кеш-пам'ять другого рівня для кешування основного об'єму ОЗП. По способу доступу до даних може бути як асинхронною так і синхронною. Тому її називають пам'ять з випадковою вибіркою (RandomAccessMemory або RAM). Кожен біт DRAM пам'яті представляєтсья в виді наявності чи відсутності зараяду на кондинсаторі. SRAM - дорожчий тип пам'яті. В якості елементарної комірки використовуєтсья статичний тригер (складається з декількох транзисторів).

Для описання характеристик швидкодії RAM використовуються цикли запису/считування даних. На запис/считування першого машинного слова використовуєтсья більше тактів, ніж на доступ до трьох наступних слів. Для асинхронної SRAM считування одного слова виконується за 3 такти, запись - 4 такти. Считування декількох слів виконується з послідовністю: 3-2-2-2 такти, а запис: 4-3-3-3. Синхронна пам'ять забеспечує доступ до даних не в випадкові моменти часу, а синхронно з тактовими імпульсами. В проміжки між ними пам'ять може готовити для доступу наступну порцію даних. В більшості материнських плат п'ятого покоління використовуєтсья різновид синхронної пам'яті - синхронно-конвеєрна SRAM (PipelinedBurstSRAM), для якої типовий час одиночної операції считування/запису складає 3 такти, а групова операція складає 3-1-1-1

такти при першому зверненні і 1-1-1-1 при наступних зверненнях,що забезпечує пришвидшення доступу більше, ніж на 25 %.

Динамічна пам'ять (DRAM) - використовуєтсья переважно в якості оперативної пам'яті загального призначення, а також пам'ять відеоадаптера. В цьому типі пам'яті біти представляються в вигляді відсутності чи наявності заряду на конденсаторі в стурктурі напівпровідникового кристалу. Кожен біт інформації записуєтсья в окремій комірці пам'яті, яка складаєтсья з конденсатора і транзистора. Якщо на конденсаторі є заряд, то це означає 1 в двійковому коді, якщо нема - 0. Транзистор при переключенні дає можливість считувати біт інформації чи записувати новий біт в пусту комірку.

* Мікросхеми динамічної пам'яті мають більшу ємність, ніж статична - для реалізації одного запам'ятовуючого елементу динамічної пам'яті потрібен 1 кондинсатор і 1 транзистор, а для запам'ятовуючого пристрою під видом тригера: 4-5 транзисторів;
* статична пам'ять працює швидше, бо комірки динамічної пам'яті не здатні зберігати заряд протягом довгого проміжку часу і потребують періодичну підзарядку-регенерацію конденсаторів. На регенерацію йде частина робочого часу мікросхеми, в ці періоди доступ до пам'яті для считування/запису закритий.
* динамічна пам'ять споживає менше енергії і суттєво дешевша.

До типів за посиланням відносяться: клас, масив, індекс, інтерфейс, стрічка.

До типів за заначенням: структура, int, byte, long ….

### Arrays

Масив – це структурована сукупність однотипних даних. Дуже зручні для пошуку. Проте в них тяжко додавати нові записи і видаляти старі. Ця операція здійснюється зі здвигом побайтно. Ця операція виконується дуже довго.

### Implementinganinterface

* Інтерфейс може бути реалізованим в одному або декількох класах;
* Можна наслідувати безліч інтерфейсів;

Implicit Implementation:

publicvoidCopyTo(Array array,int index)

{

thrownewNotImplementedException();

}

Explicit Implementation:

voidICollection.CopyTo(Array array,int index)

{

thrownewNotImplementedException();

}

Використовувати ExplicitImplementation потрібно обережно, в випадках приведення типів, наприклад чи коли 2 інтерфеси реалізую методи з однаковим ім’ям:

MyClass myClass =newMyClass();// Declared as concrete class

myclass.CopyTo//invalid with explicit

((IList)myClass).CopyTo//valid with explicit.

### Statics

Статичний елемент - це елемент, яки є загальним і не відноситься до якогось конкретного об’єкту, а швидше до типу загалом. Статичний модифікатор може бути використаний з класами, полями, методами, пропертями, опертаторами, івентами та конструкторами.

* Статичний клас не може бути ініціалізований;
* Нестатичний клас може містити статичні члени. Доступ до них відбувається через назву самого класу;
* Незалежно скільки об’єктів класу є створених, завжди існує 1 копія статичного члену;
* Статичні члени не можуть оперувати нестатичними членами з цього класу. Можуть в тому випадку якщо вони передаються параметром в метод;
* Призначенням модифікатора staticє збереження спільного значення для всіх екземплярів класу чи бажання створити лише 1 об’єкт певного класу (Singletonпаттерн);
* Сатичні методи можуть бути перевантажені, але не можуть бути overriddenоскільки вони не відносяться до вевного екземпляру класу;
* Статичні конструктори призначені для ініціалізації статичних членів та виконується перед всіма іншими конструкторами та посиланнями на статичні члени;

### Conditional Statements

**If-Else**

*If (i>0) Console.Writeline (“Positive number”)*– виводить повідомлення лише коли i>0

Console.WriteLine(i>0?"Positive number" : "Negative or Zero");

**Switch**

*For (i=0; i<10; i++)*

*{*

*Switch(i)*

*{*

*Case 0 : Console.Writeline(“Zero”); break;*

*Case 1: Console.Writeline(“One”); break;*

*Default: Console.Writeline(“Positive number”); break;*

*}*

*}*

### Loops and Iterators

**For**

Ці цикли використовуються коли відомо наперед кількість проходжень в циклі.

*For (i=0; i<10; i++){Console.WriteLine(i)}*– вихід з циклу після i=9. Останнє значення, яке буде виведено i=9.

**While**

Max=10

*inty=0;*

*while (max>y)*

*{*

*max=max-1;*

*y++;*

*Console.WriteLine("y="+y);*

*Console.WriteLine("max="+max);*

*}*

Вихід з циклу відбудеться при max = 4 &y= 6, що й буде виведене останнім на екран.

**Do-While**

Max=10

*vary=0;*

*do*

*{*

*max=max-1;*

*y++;*

*Console.WriteLine("y="+y);*

*Console.WriteLine("max="+max);*

*} while (max>=y);*

Вихід з циклу відбудеться при max = 4&y=6, що й буде виведене останнім на екран.

**Foreach**

*vararr1=new[] {1, 5, 7};*

*foreach (variinarr1)*

*{*

*Console.WriteLine(i);*

*}*

Дія виконується для кожного елементу в колекції, яка реалізовує System.Collections.IEnumerableчиSystem.Collections.Generic.IEnumerable<T> інтерфейс. Цей оператор не використовують для видалення чи додавання нових елемментів в колекцію(для цього використовують for).

**Break**

З допомогою цього оператора можна реалізувати негайний вихід з циклу, а програма продовжує роботу з наступного після циклу оператора.

*if ((num%i) ==0)*

*{*

*factor=i;*

*break;*

*}*

Вихід з циклу відбувається коли остача при діленні num на Iрівна 0.

**Continue**

З допомогою цього оператора можна реалізувати передчасне завершення кроку ітерації циклу. Тобто можна примусово перейти до наступного кроку циклу.

*for (vari=0; i<=100; i++)*

*{*

*if ((i%2) !=0) continue;*

*Console.WriteLine(i);*

*}*

Код виводить парні числа на екран. Якщо остача приділенні не рівна 0 – переходити до наступного кроку циклу.

**GoTo**

З допомогою цього оператора відбувається перехід на місце в програмі, на яке вказує цей оператор. Його вкрай нерекомендовано використовувати

### HandlingExeptions

Хендлення ексепшинів це стурктурована обробка виняткових ситуацій, які виникають в процесі роботи прорами. Наприклад підключення до неіснуючої бачи чи відкривання неіснуючого файлу, введення недозволеного формату даних… Такий підхід дозволяє стандартизовано обробляти ексепшини у всіх мовах в межах платформи та повертати досить зрозумілий опис проблеми з callstack замість простого числового коду помилки, як це було раніше. Прехоплення ексепшинів дозволяє програмі працювати далі, а не крешати при появі такого ексепшина.

- В основі лежить використання try (звернення до коду, де може виникнути виняткова ситуація), catch (для їх перехоплення), finally (вивільнення ресурсів. Цей блок виконується незалежно від того чи виник ексепшн). Ексепшини можуть бути спричинені CLR, зовнішніми бібліотеками та ключовим словом throw.

- Всі класи для хендлення ексепшинів повиніні бути унаслідувані від System.Exception.

- Ексепшини, згенеровані CLR, називаються ексепшинами на рівні системи.

- Cпеціальні типи ексешинів, які генеруться аплікацєю потрібно наслідувати відApplicationException.

### Strings

Стрічки в C# типом за посиланням .Вони є immutable, тобто незмінними. Тобто кожного разу як ми їх змінюємо (вставляємо, видаляємо) створюється новий об’єкт типу String.Також при використанні методу myString.ToUpper(), насправді ми не змінюємо поточну стрічку, а створюємо новий об’єкт.

*stringquote = "Hello Word!";*

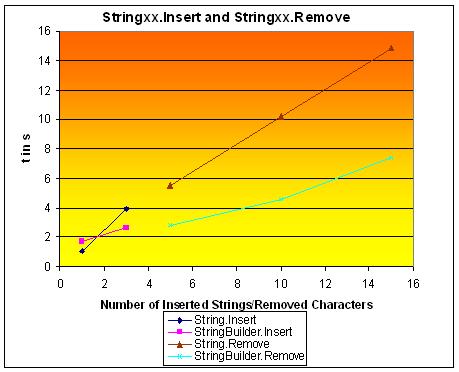
*Console.WriteLine(quote);*

*quote.Replace("Word", "Nazar");*

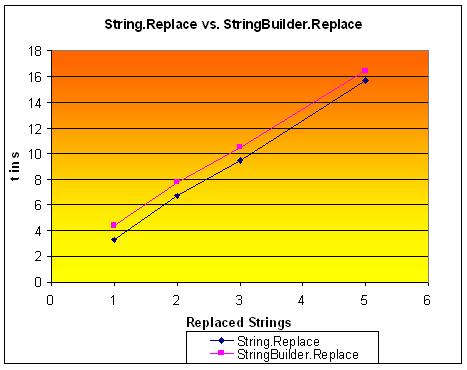
*Console.WriteLine(quote); - В цьому випадку буде виведено "HelloWord!".*

Для того щоб стрічка була змінена, потрібно хочаб присвоїти її новому об’єкту, тобто *quote = quote.Replace("Word", "Nazar");*

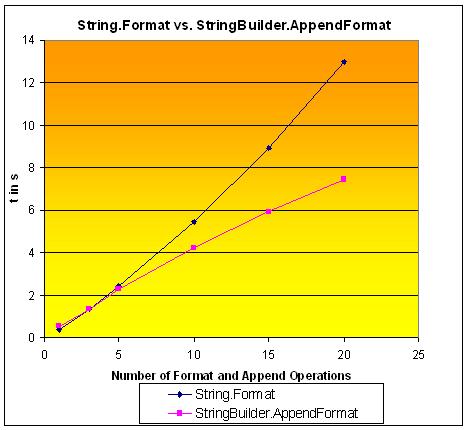
* Стрічки є nullable типом.
* Відформатовану стрічку можна отримати 2-ма спочобами: з допомогою методу String.Format(): “{індекс строкового значення: специфікатор\_формату}, value”та ToString(“специфікатор\_формату”).
* Можна задавати символи-заповнювачі спеціального формату числових даних: Console.WriteLine(“0: #,###.#”, 7844411.4). Результатом буде 7, 844, 411.4;
* При використанні оператора ‘==’, він перевантажуєтья і викликається метод ‘Equals’, який порівнює рівність вмісту двох стрічок, а не перевіряє чи вони посилаються ан один об’єкт.
* При зміні стрічки краще використовувати клас StringBuilder. Вставляння і видалення є майже завжди швидшими з використанням цього класу. Видалення є особливо швидким при кожному подвоєнні кількості видалень.



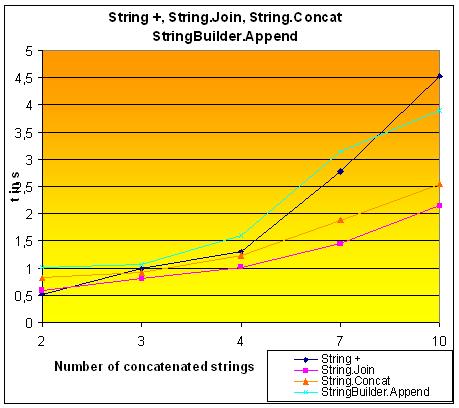
* StringBuilder є гіршим, ніж String.Replaceнавіть при великій кількості заміщень. String є швидшим приблизно на 30%.



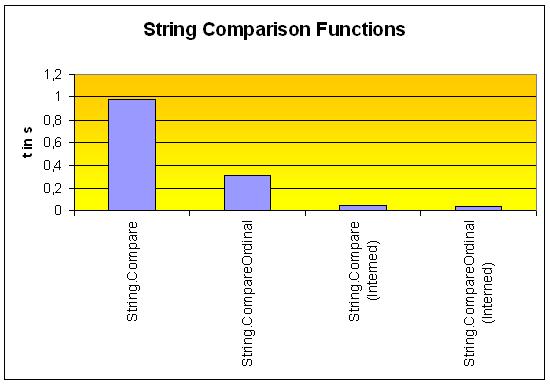
* StringBuilder.AppendFormatметод є кращим заString.Format, коли використовується "+" operator до 5 форматувань.



Найкращим вибором для конкатенації стрічок є String.Join, бо у нього реалізований кращий алгоритм.



* String.CompareOrdinalметод є дуже ефективним при порівнянні стрічок, які не виводяться на UI.



|  |  |
| --- | --- |
| **String Operation** | **Most Efficient** |
| Insert | StringBuilder.Insert > 2 Insertion Strings String.Insert otherwise |
| Remove | StringBuilder is faster > 2 characters  to remove |
| Replace | String.Replace always |
| Format | String.Format < 5 Append + Format operations StringBuilder.AppendFormat > 5 calls |
| Concatenation | + for 2 strings String.Join > 2 strings to concatenate |

### WhatisAPI

Це компонента коїсь програми з набором оперцій, вхідних і вихідних параметрів та типів, яку можна використовувати для побудови програм.

<http://en.wikipedia.org/wiki/Application_programming_interface>

### WorkingwiththeAssertionMechanism

Assert – це фунція, яка повертає true чиfalse при перевірці певного стейтменту, наприклад перевірки рівності об’єктів чи правдивості виразу.

### FileNavigationandI/O

Процес – екземпляр програми в час виконання, незалежний об’єкт, якому виділені системні ресурси. Кожен процес виконується в певному адресному просторі для того щоб один процес не міг получити доступ до даних іншого процесу. Процес складається хоча б з одного потоку. Призначення процесу –об’єднання ресурсів. Також він служить контейнером для потоків.

Потік – абстрактна послідовність інструкцій і даних загалом, прив’язана до відповідного дескриптора, яка використовується для запису і считування файлів та обміну даними між процесами.Тобто він визначає послідовність виконання коду в процесі. Будь-який потік складається з 2-ох компонент:

* Об’єкт ядра, через який ОС управляє потоком;
* Стек потоку, який містить параметри всіх функцій і локальні змінні, необхідні потоку для виконання коду.

Декілька потоків можуть можуть працювати паралельно в рамках одного процесу, використовуючи дані однієї програми. Це лежить в основі багатопоточного програмування.

System.IO – простір імен, який надає функціонал для писання і считування з файлів та пам’яті.

Класи Directoryта Fileнадають функціонал для створення, видалення, копіювання і переміщення файлів та папок з допомогою сатичних методів. Переважно повертають строкові значення.

DirectoryInfoта FileInfo– подібний функціонал, тільки реалізований через екземпляр класів. Ці класи кращі для отримання певних даних про файл чи папку, оскільки повертають строго типізовані об’єкти.

FileStream – клас для роботи з байтом чи масивом байтів. Він потрібен для кодування та розкодування інформації відповідно в потік байтів чи навпаки. Для считування чи запису символьних даних краще використовувати StreamReader таStreamWriter.

BinaryReaderта BinaryWriterдозволяє записувати та считувати файли з потоку в двійковій формі.

FileStreamWatcher – клас для відслідковування поточного стану файлу в системі.

### Dates, Numbers, andCurrency

* Форматування здійснюєтьс з допомогою 2-ох компонент: специфікатор формату та постачальник форамту. Постачальник форматувизначає порядок інтерпретації специфікатора формату. З допомогою специфікатора формату можна задавати кількість десяткових розрядів, мінімальну кількість цифт.
* Формат значень залужать від культурних і мовних особливостей локалізації програмного забезпечення.

### Parsing, Tokenizing, andFormatting

Parsing – це процес считування і перетворення якогось коду в потрібний формат, витягуючи з нього потрібну інформацію. Наприклад це считування htmlкоду і повернення набору тегів і тексту.

Tokenizing – це ділення строки на менші елементи. Наприклад використання RegEx.Split чи String.Split.

Formatting – це побудова строки за певним шаблоном.

### Collections: Arrays, Lists, Dictionaries

Колекції – це сукупність погрупованих даних. Основною їх перевагою є те, що вони стандартизують обробку груп об’єктів в програмі.

Array – це структурована сукупність однотипних даних з заздалегідь визначеною кількістю. Дуже зручні для пошуку. Проте в них тяжко додавати нові записи і видаляти старі. Ця операція здійснюється зі здвигом побайтно. Ця операція виконується дуже довго. Робота з ними є швидшою, ніж з іншими типами колекцій.

List – строго типізований список об’єктів, доступних по індексу. Їх добре використвувати коли наперед невідома кількість елементів всередині, та потрібно додавати чи видаляти елементи. Пошук по списку є доволі довгим, оскільки алгоритм по елементах проходить один за одним.

Dictionary–це реалізація хеш-таблиці, тобто складається з пари ключ-значення. Але основною відмінністю є те, що словники є дженерік, тобто можуть сторити різнотипні дані. За рахунок хеш-таблиці в основі алгоритм пошуку в словниках є ефективнішим, ніж в списках.

Ефективнішим для зберігання є тип Tuple.

### Serialization

Серіалізація – це процес збереження поточного стану об’єкту в потоці через конвертування його в потік байтів. Ця інформація потім використовується для подальшого обробки та збереження в базі, пам’яті чи файлі. Зворотній процес називається десеріалізацією. Коли об’єкт серіалізується, то всі асоційовані з ним дані, тобто дані його класу теж серіалізуються. Якщо серіалізується об’єкт класу-наслідника, то батьківський клас теж серіалізується. Якщо ця структура є великою, то вони утворюють граф об’єктів.

### StaticNestedClasses

Тип, який є оголошений в середині класу чи структури є вкладеним. Вони здебільшого використовуються для групування коду всередині дуже великого класу. Статичні вкладені класи створюють для їх використання ззовні через нестатичні класи.

### Annotations

Атрибути – це засіб зв’язування метаданих з кодом в рантаймі з допомогою рефлексії:

* Вони додають в програму метадані;
* Вони пожуть приймати аргументи;
* Програма може провірити метадані чи метедані інших програм з допомогою рефлексії.

Прикладом використання атрибутів є атрибути для серіалізації об’єктів чи считування даних з зовнішніх файлів.

### Locating Files and Directories Using Paths

Локалізація щляку до файлу за іменем з допомогою методу Path.GetFullPath, задавши назву файлу параметром. Цей метод дозволяє працювати з файлами незалежно від платформи без хардкоду.

FileStreamWatcher – клас для відслідковування поточного стану файлу в системі за допомогою прописання шляху до файла чи папки.

### Obtaining and Managing Files and Directories

Класи Directoryта Fileнадають функціонал для створення, видалення, копіювання і переміщення файлів та папок з допомогою сатичних методів. Переважно повертають строкові значення.

DirectoryInfoта FileInfo– подібний функціонал, тільки реалізований через екземпляр класів. Ці класи кращі для отримання певних даних про файл чи папку, оскільки повертають строго типізовані об’єкти.

### StreamIO

System.IO – простір імен, який надає функціонал для писання і считування з файлів та пам’яті.

## Refactoring

### Refactoring Concept (what/when/why)

Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. It is a disciplined way to clean up code that minimizes the chances of introducing bugs. In essence when you refactor you are improving the design of the code after it has been written.

Refactor When You Add Function

Refactor When You Need to Fix a Bug

Refactor As You Do a Code Review

We need to refactor code because of for main problems that occurs while developing software:

* Programs that are hard to read are hard to modify.
* Programs that have duplicated logic are hard to modify.
* Programs that require additional behavior that requires you to change running code are hard to modify.
* Programs with complex conditional logic are hard to modify.

### Smells Catalog and possible re-factorings

Duplicated Code

Long Method

Large Class

Long Parameter List

Divergent Change

Shotgun Surgery

Feature Envy

Data Clumps

Primitive Obsession

Switch Statements

Parallel Inheritance Hierarchies

Lazy Class

Speculative Generality

Temporary Field

Message Chains

Middle Man

Inappropriate Intimacy

Alternative Classes with Different Interfaces

Incomplete Library Class

Data Class

Comments

### Moving Features Between Objects (basic)

One of the most fundamental, if not the fundamental, decision in object design is deciding where to put responsibilities. I've been working with objects for more than a decade, but I still never get it right the first time. That used to bother me, but now I realize that I can use refactoring to change my mind in these cases.

Move Method

A method is, or will be, using or used by more features of another class than the class on which it is defined. Create a new method with a similar body in the class it uses most. Either turn the old method into a simple delegation, or remove it altogether.

Move Field

A field is, or will be, used by another class more than the class on which it is defined. Create a new field in the target class, and change all its users.

### Organizing Data (basic)

One of the useful things about object languages is that they allow you to define new types that go beyond what can be done with the simple data types of traditional languages. It takes a while to get used to how to do this, however. Often you start with a simple data value and then realize that an object would be more useful. Replace Data Value with Object allows you to turn dumb data into articulate objects. When you realize that these objects are instances that will be needed in many parts of the program, you can use Change Value to Reference to make them into reference objects.

Encapsulate Field

You are accessing a field directly, but the coupling to the field is becoming awkward.Create getting and setting methods for the field and use onlythose to access the field.

There is a public field. Make it private and provide accessors.

Encapsulate Collection

A method returns a collection. Make it return a read-only view and provide add/remove methods

### Composing Methods (basic)

A large part of my refactoring is composing methods to package code properly. Almost all the time the problems come from methods that are too long. Long methods are troublesome because they often contain lots of information, which gets buried by the complex logic that usually gets dragged in. The key refactoring is Extract Method, which takes a clump of code and turns it into its own method. Inline Method is essentially the opposite. You take a method call and replace it with the body of the code. I need Inline Method when I've done multiple extractions and realize some of the resulting methods are no longer pulling their weight or if I need to reorganize the way I've broken down methods.

Extract Method

You have a code fragment that can be grouped together. Turn the fragment into a method whose name explains the purpose of the method.

Inline Method

A method's body is just as clear as its name. Put the method's body into the body of its callers and remove the method.

Inline Temp

You have a temp that is assigned to once with a simple expression, and the temp is getting in the way of other refactorings. Replace all references to that temp with the expression

Replace Temp with Query

You are using a temporary variable to hold the result of an expression.

Extract the expression into a method. Replace all references to the temp with the expression. The new method can then be used in other methods.

Split Temporary Variable

You have a temporary variable assigned to more than once, but is not a loop variable nor a collecting temporary variable.

Make a separate temporary variable for each assignment.

### Simplifying Conditional Expressions (basic)

Conditional logic has a way of getting tricky, so here are a number of refactorings you can use to simplify it. The core refactoring here is Decompose Conditional, which entails breaking a conditional into pieces. It is important because it separates the switching logic from the details of what happens. The other refactorings in this chapter involve other important cases. Use Consolidate Conditional Expression when you have several tests and all have the same effect. Use Consolidate Duplicate Conditional Fragments to remove any duplication within the conditional code.

**Decompose Conditional Expression**

You have a complicated conditional (if -then-else) statement. Extract methods from the condition, then part, and else parts.

**Consolidate Conditional Expression**

You have a sequence of conditional tests with the same result. Combine them into a single conditional expression and extract it.

**Consolidate Duplicate Conditional Fragments**

The same fragment of code is in all branches of a conditional expression. Move it outside of the expression.

**Remove Control Flag**

You have a variable that is acting as a control flag for a series of boolean expressions.

Use a break or return instead.

**Replace Conditional with Polymorphism**

You have a conditional that chooses different behavior depending on the type of an object.

Move each leg of the conditional to an overriding method in a subclass. Make the original method abstract.

### Making Method Calls Simpler

Objects are all about interfaces. Coming up with interfaces that are easy to understand and use is a key skill in developing good object-oriented software. This chapter explores refactorings that make interfaces more straightforward. Often the simplest and most important thing you can do is to change the name of a method. Naming is a key tool in communication. If you understand what a program is doing, you should not be afraid to use Rename Method to pass on that knowledge. You can (and should) also rename variables and classes. On the whole these renamings are fairly simple text replacements, so I haven't added extra refactorings for them. Parameters themselves have quite a role to play with interfaces. Add Parameter and Remove Parameter are common refactorings. Programmers new to objects often use long parameter lists, which are typical of other development environments. Objects allow you to keep parameter lists short, and several more involved refactorings give you ways to shorten them. If you are passing several values from an object, use Preserve Whole Object to reduce all the values to a single object. If this object does not exist, you can create it with Introduce Parameter Object. If you can get the data from an object to which the method already has access, you can eliminate parameters with Replace Parameter with Method. If you have parameters that are used to determine conditional behavior, you can use Replace Parameter with Explicit Methods. You can combine several similar methods by adding a parameter with Parameterize Method.

### Dealing with Generalization

Generalization produces its own batch of refactorings, mostly dealing with moving methods around a hierarchy of inheritance. Pull Up Field and Pull Up Method both promote function up a hierarchy, and Push Down Method and Push Down Field push function downward. Constructors are a little bit more awkward to pull up, so Pull Up Constructor Body deals with those issues. Rather than pushing down a constructor, it is often useful to use Replace Constructor with Factory Method. If you have methods that have a similar outline body but vary in details, you can use Form Template Method to separate the differences from the similarities. In addition to moving function around a hierarchy, you can change the hierarchy by creating new classes. Extract Subclass, Extract Superclass, and Extract Interface all do this by forming new elements out of various points. Extract Interface is particularly important when you want to mark a small amount of function for the type system. If you find yourself with unnecessary classes in your hierarchy, you can use Collapse Hierarchy to remove them. Sometimes you find that inheritance is not the best way of handling a situation and that you need delegation instead. Replace Inheritance with Delegation helps make this change. Sometimes life is the other way around and you have to use Replace Delegation with Inheritance.

### Creation

**Replace Constructors with Creation Methods**

Constructors on a class make it hard to decide which constructor to call during development. Replace the constructors with intention-revealing Creation Methods that return object instances.

**Move Creation Knowledge to Factory**

Data and code used to instantiate a class is sprawled across numerous classes. Move the creation knowledge into a single Factory class.

<http://www.informit.com/articles/article.aspx?p=1398606&seqNum=2>

**Encapsulate Classes with Factory**

Clients directly instantiate classes that reside in one package and implement a common interface. Make the class constructors non-public and let clients create instances of them using superclass Creation Methods.

**Encapsulate Composite with Builder**

Building a Composite is repetitive, complicated, or error-prone. Simplify the build by letting a Builder handle the details.

### Simplification

**Replace Conditional Logic with Strategy**

Conditional logic in a method controls which of several variants of a calculation are executed. Create a Strategy for each variant and make the method delegate the calculation to a Strategy instance

**Replace State-Altering Conditionals with State**

The conditional expressions that control an object's state transitions are complex. Replace the conditionals with State classes that handle specific states and transitions between them.

**Replace Implicit Tree with Composite**

You implicitly form a tree structure, using a primitive representation, such as a String. Replace your primitive representation with a Composite.

**Replace Conditional Dispatcher with Command**

Conditional logic is used to dispatch requests and execute actions. Create a Command for each action. Store the Commands in a collection and replace the conditional logic with code to fetch and execute Commands.

### Generalization

**Form Template Method**

Your subclasses implement algorithms that contain similar steps in the same order.Move the algorithm structure and identical steps to a subclass, and leave implementation of the different steps in the subclasses.

<https://refactoring.guru/form-template-method>

**Extract Composite**

Subclasses in a hierarchy implement the same Composite.Extract a superclass that implements the Composite.

**Replace Hard-Coded Notifications with Observer**

Subclasses are hard-coded to notify a single instance of another class.

Remove the subclasses by making their superclass capable of notifying one or more instances of any class that implements an Observer interface.

**Red green refactoring**

Improve the structure of the code to ease future changes and maintenance.

* Does repeated code exist in the current class?
* Is the name of the class under test appropriate?
* Do the public and protected method names describe their intent? Are they readable? Rename refactorings are between the most powerful ones.
* Does repeated code exist in different classes? Is there a missing domain concept? You can extract abstract classes or refactor towards composition. At this high-level the refactoring should be also applied to the unit tests, and there are many orthogonal techniques you can apply so I won't describe them all here.

## Debugging, troubleshooting and defect fixing

### Logging to text files / console output for debugging

* Для виведення меседжу в аутпут секцію використовувати методSystem.Diagnostics.Debug.Write();
* Для запису логів в текстовий файл:
  + Створити змінну зі шляхом до лог файлу;
  + Створити об’єкт класу StreamWriter та заюзати методAppendText() з вказаним шляхом до логу;
  + Через String.Formatпрописати форматування логу;
  + Через метод StreamWriter.Writelineписати логи.

- Для перенапрвлення консольних повідомлень юзати Console.SetOut(об’єкт StreamWriter);

* Для виведення дебаг інформації в консоль юзати TextWriterTraceListener

TextWriterTraceListener tr1 = new TextWriterTraceListener(System.Console.Out);

Debug.Listeners.Add(tr1);

* Для запису дебаг інформації в файл також

TextWriterTraceListener tr2 = new TextWriterTraceListener(System.IO.File.CreateText("Output.txt"));

Debug.Listeners.Add(tr2);

### Using breakpoints for inspecting automation code

Дебагінг допомагає зрозуміти що робить програма в рантаймі. Брейкпоінт – точка, яка вказує на місце, в якому потрібно зупинитись та яке потрібно дослідити.

## Managing versions

### Fundamental concepts: revisions, working copy, repository

Subversion uses URLs to identify versioned files and directories in Subversion repositories. For the most part, these URLs use the standard syntax, allowing for server names and port numbers to be specified as part of the URL.

A Subversion working copy is an ordinary directory tree on your local system, containing a collection of files. You can edit these files however you wish, and if they're source code files, you can compile your program from them in the usual way. Your working copy is your own private work area: Subversion will never incorporate other people's changes, nor make your own changes available to others, until you explicitly tell it to do so. You can even have multiple working copies of the same project.

Each time the repository accepts a commit, this creates a new state of the filesystem tree, called a revision. Each revision is assigned a unique natural number, one greater than the number of the previous revision. The initial revision of a freshly created repository is numbered 0 and consists of nothing but an empty root directory.

### Getting (checking out) data from repository

When working on a project with a team, you'll want to update your working copy to receive any changes other developers on the project have made since your last update. Use **svn update** to bring your working copy into sync with the latest revision in the repository

### Examining local changes, comparing working copy to repository

Once you've finished making changes, you need to commit them to the repository, but before you do so, it's usually a good idea to take a look at exactly what you've changed. By examining your changes before you commit, you can make a more accurate log message. You may also discover that you've inadvertently changed a file, and this gives you a chance to revert those changes before committing. Additionally, this is a good opportunity to review and scrutinize changes before publishing them. You can see an overview of the changes you've made by using svn status, and dig into the details of those changes by using svn diff.

### Updating working copy, merging changes

When working on a project with a team, you'll want to update your working copy to receive any changes other developers on the project have made since your last update. Use svn update to bring your working copy into sync with the latest revision in the repository.

### Updating to specific version

In addition to all of the previous commands, you can use svn update and svn checkout with the --revision option to take an entire working copy “back in time”.

### Examining change history

We've already seen svn diff before—it displays file differences in unified diff format; we used it to show the local modifications made to our working copy before committing to the repository.

In fact, it turns out that there are three distinct uses of svn diff:

* Examining local changes
* Comparing your working copy to the repository
* Comparing repository revisions

### Discarding changes

svn revert item has exactly the same effect as deleting item from your working copy and then running svn update -r BASE item. However, if you're reverting a file, svn revert has one very noticeable difference—it doesn't have to communicate with the repository to restore your file.

### Resolving conflicts in code

Merging conflicts by hand can be quite intimidating the first time you attempt it, but with a little practice, it can become as easy as falling off a bike.

### Copying, moving files, folders in repository

The easiest way to copy files and folders from within a working copy is to use the right drag menu. When you right drag a file or folder from one working copy to another, or even within the same folder, a context menu appears when you release the mouse.Now you can copy existing versioned content to a new location, possibly renaming it at the same time.

### Committing changes

The svn commit command sends all of your changes to the repository. When you commit a change, you need to supply a log message describing your change. Your log message will be attached to the new revision you create. If your log message is brief, you may wish to supply it on the command line using the --message (or -m) option.

### Creating, discovering and breaking locks (if supported)

A repository lock isn't sacred—in Subversion's default configuration state, locks can be released not only by the person who created them, but by anyone. When somebody other than the original lock creator destroys a lock, we refer to this as breaking the lock.

From the administrator's chair, it's simple to break locks. The svnlook and svnadmin programs have the ability to display and remove locks directly from the repository.

The more interesting option is to allow users to break each other's locks over the network. To do this, Sally simply needs to pass the --force to the svn unlock command.

### Importing sources into repository

Although it does the job, it has several shortcomings:

* There is no way to select files and folders to include, aside from using the global ignore settings.
* The folder imported does not become a working copy. You have to do a checkout to copy the files back from the server.
* It is easy to import to the wrong folder level in the repository.

Before you import your project into a repository you should:

* Remove all files which are not needed to build the project (temporary files, files which are generated by a compiler e.g. \*.obj, compiled binaries, ...)
* Organize the files in folders and sub-folders. Although it is possible to rename/move files later it is highly recommended to get your project's structure straight before importing!

### Exporting sources from repository

Sometimes you may want a clean copy of your working tree without the .svn directory, e.g. to create a zipped tarball of your source, or to export to a web server. Instead of making a copy and then deleting the .svn directory manually, TortoiseSVN offers the command TortoiseSVN → Export.... Exporting from a URL and exporting from a working copy are treated slightly differently.

If you execute this command on an unversioned folder, TortoiseSVN will assume that the selected folder is the target, and open a dialog for you to enter the URL and revision to export from. This dialog has options to export only the top level folder, to omit external references, and to override the line end style for files which have the svn:eol-style property set.

### Comparing repository revisions

If you want to see the difference between two revisions which are already committed, use the Revision Log dialog and select the two revisions you want to compare (using the usual Ctrl-modifier). Then select Compare revisions from the context menu.

If you did this from the revision log for a folder, a Compare Revisions dialog appears, showing a list of changed files in that folder.

### Creating a branch

Select the folder in your working copy which you want to copy to a branch or tag, then select the command TortoiseSVN → Branch/Tag....

The default destination URL for the new branch will be the source URL on which your working copy is based. You will need to edit that URL to the new path for your branch/tag.

### Working with branch

To be able to work with your freshly generated branch or tag you have several ways to handle it. You can:

* TortoiseSVN → Checkout to make a fresh checkout in an empty folder. You can check out to any location on your local disk and you can create as many working copies from your repository as you like.
* Switch your current working copy to the newly created copy in the repository. Again select the top level folder of your project and use TortoiseSVN → Switch... from the context menu.
* In the next dialog enter the URL of the branch you just created. Select the Head Revision radio button and click on OK. Your working copy is switched to the new branch/tag.
* Switch works just like Update in that it never discards your local changes. Any changes you have made to your working copy which have not yet been committed will be merged when you do the Switch. If you do not want this to happen then you must either commit the changes before switching, or revert your working copy to an already-committed revision (typically HEAD).
* If you want to work on trunk and branch, but don't want the expense of a fresh checkout, you can use Windows Explorer to make a copy of your trunk checkout in another folder, then TortoiseSVN → Switch... that copy to your new branch.

### Traversing branches

The svn switch command transforms an existing working copy to reflect a different branch. While this command isn't strictly necessary for working with branches, it provides a nice shortcut. In our earlier example, after creating your private branch, you checked out a fresh working copy of the new repository directory.

### Common branching patterns (rebasing in Git)

Main brunching patterns:

* Release Branches
* Feature Branches

### Merging branches

For projects that have a large number of contributors, it's common for most people to have working copies of the trunk. Whenever someone needs to make a long-running change that is likely to disrupt the trunk, a standard procedure is to create a private branch and commit changes there until all the work is complete.In Subversion terminology, the general act of replicating changes from one branch to another is called merging, and it is performed using various invocations of the svn merge subcommand.

### Tagging (labeling in TFS)

Another common version control concept is a tag. A tag is just a “snapshot” of a project in time. In Subversion, this idea already seems to be everywhere. Each repository revision is exactly that—a snapshot of the filesystem after each commit.

### Creating and applying patches (shelvesets in TFS)

First you need to make and test your changes. Then instead of using TortoiseSVN → Commit... on the parent folder, you select TortoiseSVN → Create Patch...

you can now select the files you want included in the patch, just as you would with a full commit. This will produce a single file containing a summary of all the changes you have made to the selected files since the last update from the repository.

The columns in this dialog can be customized in the same way as the columns in the Check for modifications dialog. Read the section called “Local and Remote Status” for further details.

By clicking on the Options button you can specify how the patch is created. For example you can specify that changes in line endings or whitespaces are not included in the final patch file.

You can produce separate patches containing changes to different sets of files. Of course, if you create a patch file, make some more changes to the same files and then create another patch, the second patch file will include both sets of changes.

### Revision specifiers

The Subversion client understands a number of revision keywords. These keywords can be used instead of integer arguments to the --revision (-r) option, and are resolved into specific revision numbers by Subversion:

* HEAD - The latest (or “youngest”) revision in the repository.
* BASE - The revision number of an item in a working copy. If the item has been locally modified, this refers to the way the item appears without those local modifications.
* COMMITTED - The most recent revision prior to, or equal to, BASE, in which an item changed.
* PREV - The revision immediately before the last revision in which an item changed. Technically, this boils down to COMMITTED-1.

### Data recovery

You can create dump from svn repository and restore data from it.

### Using external editors and merge tools

Subversion calls external diff programs with parameters suitable for the GNU diff utility, and expects only that the external program will return with a successful error code. For most alternative diff programs, only the sixth and seventh arguments—the paths of the files that represent the left and right sides of the diff, respectively—are of interest. Note that Subversion runs the diff program once per modified file covered by the Subversion operation, so if your program runs in an asynchronous fashion (or is “backgrounded”), you might have several instances of it all running simultaneously. Finally, Subversion expects that your program return an error code of 1 if your program detected differences, or 0 if it did not—any other error code is considered a fatal error.

### Distributed VCS (if supported)

Supported in GIT

### Automated build concept

Continuous integration (CI) is the practice, in software engineering, of merging all developer working copies with a shared mainline several times a day.Later elaborations of the concept introduced build servers, which automatically run the unit tests periodically or even after every commit and report the results to the developers.

<http://en.wikipedia.org/wiki/Continuous_integration>

### Integrating CI and version control

In Jenkins this type of integration is made using various plugins.

### Building test scripts with the CI process

Test scripts can be divided on some categories. Some smote tests be ran during after commit build to make sure that all basic functionality is working fine.

### Executing CI scripts

There are two ways to run CI scripts. First one is to trigger build manually and second is to configure some riggers to start build automatically.

### Monitoring build process

In Jenkins it is possible to monitor build process with help of console output functions.

### Build status reporting (notification)

Jenkins have a lot of various plugins for informing about build status. Also there is a custom possibility to send e-mails after build actions are complete.

### Configuring user’s authorization

You can implement user authorization on project level and site level. For detailed information visit Jenkins wiki.

<https://wiki.jenkins-ci.org/display/JENKINS/Authentication>

### Configuration reports publication and auto-email notifications

Also can be done in post-build section with help of different plugins.

### How to configure and launch UI tests on remote slave via CI

First it is important to start slave and connect it to master. After connection is successfully established it is possible to configure Jenkins to run some jobs on this slave.

# Standard Automation Components

## CSV format

A comma-separated values (CSV) (also sometimes called character-separated values) file stores tabular data (numbers and text) in plain-text form. Plain text means that the file is a sequence of characters, with no data that has to be interpreted as binary numbers. A CSV file consists of any number of records, separated by line breaks of some kind; each record consists of fields, separated by some other character or string, most commonly a literal comma or tab. Usually, all records have an identical sequence of fields.

<http://en.wikipedia.org/wiki/Comma-separated_values>

## XML format

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format which is both human-readable and machine-readable. It is defined by the W3C's XML 1.0 Specification and by several other related specifications, all of which are free open standards.

<http://en.wikipedia.org/wiki/XML>

## Parsing XML

In C# there are two classes that are designed to parse XML documents:

* XmlDocument Class
* XDocument Class

<https://msdn.microsoft.com/en-us/library/system.xml.xmldocument.aspx>

<https://msdn.microsoft.com/en-us/library/system.xml.linq.xdocument.aspx>

## Forming XML outputs

XmlWriter writes XML data from objects in memory.

<https://msdn.microsoft.com/en-us/library/System.Xml.XmlWriter(v=vs.110).aspx>

## Forming HTML outputs

For writing HTML file use HtmlTextWriter Class.

<https://msdn.microsoft.com/en-us/library/system.web.ui.htmltextwriter(v=vs.110).aspx>

## Creating separate test and test suite reports

MSTest creates .trx test report file. It uses XML inside so it is easy to parse it and form some special reports if needed.

## Encrypting/ decrypting sensitive data

According to HIPPA standards all sensitive data should be encrypted. There are a lot of encryption algorithms that can be applied in application so I think that detailed realization of those algorithms is not so important.

## Data driven testing

Data-driven testing (DDT) is a term used in the testing of computer software to describe testing done using a table of conditions directly as test inputs and verifiable outputs as well as the process where test environment settings and control are not hard-coded. In the simplest form the tester supplies the inputs from a row in the table and expects the outputs which occur in the same row. The table typically contains values which correspond to boundary or partition input spaces. In the control methodology, test configuration is "read" from a database.

<http://en.wikipedia.org/wiki/Data-driven_testing>

## RDBMS

A relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as invented by E. F. Codd, of IBM's San Jose Research Laboratory. Many popular databases currently in use are based on the relational database model.

<http://en.wikipedia.org/wiki/Relational_database_management_system>

## Database connection

To connect to database it is required to create SqlConnection instance and provide connection string for it. Basically connection sting looks like this SqlConnection myConnection = new SqlConnection("user id=username;password=password;server=serverurl;Trusted\_Connection=yes;database=database;connection timeout=30");

After you finished working with database don’t forget to close connection.

<http://www.codeproject.com/Articles/4416/Beginners-guide-to-accessing-SQL-Server-through-C>

## Writing SQL queries to retrieve test data

There are 2 ways to write sql queries for retrieving data from database

* Using SqlCommand class
* Using Entity framework and LINQ and Lambda expressions

## Writing SQL queries to post test results data

The same as above

## SQL Lite database

SQLite is a relational database management system contained in a C programming library. In contrast to other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.

<http://en.wikipedia.org/wiki/SQLite>

## Posting test results via e-mail

I know two possible ways to send test results via e-mail. First one is to configure CI system to send e mail notification in post-build actions. Second is to do it using MailMessage and SmtpClient classes.

## Testing SQL database storages

Stupid question

<http://en.wikipedia.org/wiki/Database_testing>

## Testing SQL business logic (stored procedures)

The same as above

# Web GUI Testing

## WebDriver concept

Selenium is a portable software testing framework for web applications. Selenium provides a record/playback tool for authoring tests without learning a test scripting language (Selenium IDE). It also provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including Java, C#, Groovy, Perl, PHP, Python and Ruby. The tests can then be run against most modern web browsers. Selenium deploys on Windows, Linux, and Macintosh platforms. It is open-source software, released under the Apache 2.0 license, and can be downloaded and used without charge.

<http://en.wikipedia.org/wiki/Selenium_(software)>

## Auto-generated IDs

There are few ways to handle the problem of auto-generated IDs

* Absolute XPath
* Closest stable element
* Stable preceding text
* Text content

<http://www.optimusinfo.com/dynamic-ids-headache-automation-great-challenge/>

## Multiple browser support

Selenium supports multiple browser testing. For using cross-browser feature you have to install all required additional software and instantiate driver of exact browser type that is needed.

## xPath selectors

XPath, the XML Path Language, is a query language for selecting nodes from an XML document.

PROS:

Allows very precise locators

CONS:

Slower than CSS

Relies on browser’s XPath implementation which is not always complete (especially on IE) and as such is not recommended for cross-browser testing

<http://hedleyproctor.com/2011/05/tutorial-writing-xpath-selectors-for-selenium-tests/>

<http://www.silverwareconsulting.com/index.cfm/2013/3/8/Very-nice-xpath-and-css-selector-locator-cheat-sheet-for-Selenium>

## css Selectors

CSS selectors allow you to select and manipulate HTML elements. CSS selectors are used to "find" (or select) HTML elements based on their id, class, type, attribute, and more.

PROS:

Much faster than XPath

Widely used

Provides a good balance between structure and attributes

Allows for selection of elements by their surrounding context

CONS:

They tend to be more complex and require a steeper learning curve

<http://seleniumeasy.com/selenium-tutorials/css-selectors-tutorial-for-selenium-with-examples>

## ID selectors

The Id strategy looks for an element in the page having an id attribute corresponding to the specified pattern. <label id="my\_id" /> will be matched by a locator like id=my\_id or just my\_id

PROS:

Each id is supposed to be unique so no chance of matching several elements

CONS:

Works well only on elements with fixed ids and not generated ones

<https://newcircle.com/static/bookshelf/selenium_tutorial/locators.html>

## Handling Ajax Requests

It is recommended to use WebDriverWait class for handling Ajax requests.

<http://blog.devscrum.net/2011/04/selenium-2-webdriver-handling-ajax-page-requests/>

## JavaScript execution using WebDriver interface

You can execute arbitrary javascript to find an element and as long as you return a DOM Element, it will be automatically converted to a WebElement object.

## Load testing Web UI

To perform load testing it is possible to run selenium tests in different threads. To spread the load of testing across several machines you can use Selenium Grid.

## Measuring performance of the user interface

Простий метод – стопвоч тест за допомогою звичайного годинника + браузер дев тулс. Складніший – написання функцій, які будуть трекати стан завантаження сторінки.

## RESTvsOthers

На даний момент популярні 3 способи реалізації веб-сервісів: XML-RPC, SOAP, REST.

SOA – сервіс-орієнтована архітектура, яка базується на взаємодії окремих сервісів з використанням протоколів, наприклад SOAPчи RPC/XML.

WSDL (WebServicesDescriptionLanguage) – мова опису веб сервісів з допомогою xmlформату. WSDLфайл надає зрозумілу для машини інструкцію того як сервіс може бути визваний і де він знаходиться, які параметри очікує, і які дані повертає.

XML- RPC– это «remoteprocedurecall», понятие очень старое, объединяющие древние, средние и современные протоколы, которые позволяют вызвать метод в другом приложении. XML-RPC – это протокол, появившийся в 1998, вскоре после появления XML. Изначально поддерживался Microsoft, но вскоре Microsoft полностью переключилась на SOAP и поэтому в .NetFramework мы не найдем классов для поддержки этого протокола. Несмотря на это, XML-RPC продолжает жить до сих пор в различных языках (особенно в PHP), видимо, заслужил любовь разработчиков своей простотой.

SOAP – протокол обміну структурованими даними. Він сумісний з архітектурою WS-\*, яка може використовувати і інші формати, такі якbinaryxmlта використовувати різні транспорти: http, tcp/ip.SOAP-сервіси прив’язані до xmlформату. Також у них стандартизована обробка помилок. Повільніші за рестфул сервіси, проте надійніші в плані безпеки і закінченості транзакцій, а також дозволяють реалізовувати складні сценарії роботи з об’єктами. Їх здебільшого використовують для фінансових транзакцій, де швидкість грає не основну роль.

REST – ресурсо-орієнтований архітектурний стиль, в якому ресурси надаються сервісами. Над ресурсами можна здійснювати CRUDоперації.В основі лежитьHTTPпротокол. Обробка помилок здійснюється з допомогою HTTPкодів.Не має підтримки WSDL, тому нема можливості швидко згенерувати проксі для клієнта, тобто httpметоди потрібно самому реалізовувати через HttpWebClient.Для серіалізації даних використовуються різні формати. У цього стилю є свої принципи:

* Give every “thing” an ID.  
  Очччень желательно.
* Link things together.  
  Например, в страницу (представление) о Mercedes C218 хорошо бы добавить ссылку на страницу конкретно о двигателе данной модели, чтобы желающие могли сразу туда перейти, а не тратить время на поиск этой самой страницы.
* Use standard methods.  
  Имеется в виду, экономьте свои силы и деньги заказчика, используйте стандартные методы HTTP, например GET

http://www.example.com/cars/00345

для получения данных вместо определения собственных методов вроде getCar?id=00345.

* Resources can have multiple representations.  
  Одни и те же данные можно вернуть в XML или JSON для программной обработки или обернутыми в красивый дизайн для просмотра человеком.
* Communicate statelessly.  
  Да, RESTful сервис должен быть как идеальный суд – его не должно интересовать ни прошлое подсудимого (клиента), ни будущее – он просто выносит приговор (отвечает на запрос).

1. **SOAP – это целое семейство протоколов и стандартов**, откуда напрямую вытекает, что это более тяжеловесный и сложный вариант с точки зрения машинной обработки. Поэтому REST работает быстрее.
2. SOAP используют HTTP как транспортный протокол, в то время как REST базируется на нем. Это означает, что все существующие наработки на базе протокола HTTP, такие как кеширование на уровне сервера, масштабирование, продолжают так же работать в REST архитектуре, а для SOAP необходимо искать другие средства. Взамен этого SOAP сервисы получают такое мифическое свойство, как возможность работать с любым протоколом транспортного уровня вместо HTTP, однако практической пользы от него зачастую не больше, чем сотрудникам [Челябинского трубопрокатного завода](http://ru.wikipedia.org/wiki/%D0%A7%D0%B5%D0%BB%D1%8F%D0%B1%D0%B8%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D1%80%D1%83%D0%B1%D0%BE%D0%BF%D1%80%D0%BE%D0%BA%D0%B0%D1%82%D0%BD%D1%8B%D0%B9_%D0%B7%D0%B0%D0%B2%D0%BE%D0%B4) от большого количесва статей в википедиях на [мертвых языках](http://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%80%D1%82%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA).
3. Есть [мнение](http://ajaxonomy.com/2008/xml/web-services-part-1-soap-vs-rest), что разработка RESTful сервисов намного проще. Наверное, это правда, если использовать Notepad в качестве основной среды разработки, но вот с использованием наших чудесных средств разработки, я позволю себе усомниться в верности этого утверждения.
4. В первом гугловском [результате](http://spf13.com/post/soap-vs-rest) по запросу «RESTvsSOAP» акцентируется внимание на том, что ответ REST может быть представлен в различных форматах, а SOAP привязан к XML. Это действительно важный фактор, достаточно представить себе вызов сервиса из javascript, ответ на который мы определенно хотим получать в JSON.
5. **«RESTvsSOAP» можно перефразировать в «Простота vs Стандарты»**, что проявляется в том, что для SOAP мы имеем протокол WSDL для исчерпывающего описания веб-сервиса, который с использованием все тех же чудесных средств разработки прото-таки волшебным образом делает почти всю работу за нас. Со стороны REST мы имеем загадочный и неиспользуемый протокол WADL, который, в принципе, и не нужен – он мешает простоте.
6. Второй аспект предыдущего пункта – обработка ошибок. В SOAP она полностью стандартизована, а REST может использовать давно известные коды ошибок HTTP (если здесь Вас посетила мысль, что это же очевидно и зачем я это пишу, то значит Вы внимательно читаете статью).
7. То, с чего можно было бы начать, но я припас напоследок. Это одна из ключевых мыслей. **SOAP работает с операциями, а REST – с ресурсами**. Этот факт в совокупности с отсутствием клиентского состояния у RESTful сервисов приводит нас к тому, что такие вещи как транзакции или другая сложная логика должна реализовываться «SOAP-но».

 SOAP активно использует XML для кодирования запросов и ответов, а также строгую типизацию данных, гарантирующую их целостность при передаче между клиентом и сервером. С другой стороны, запросы и ответы в REST могут передаваться в ASCII, XML, JSON или любых других форматах, распознаваемых одновременно и клиентом, и сервером. Кроме того, в модели REST отсутствуют встроенные требования к типизации данных. В результате пакеты запросов и ответов в REST имеют намного меньшие размеры, чем соответствующие им пакеты SOAP.

 В модели SOAP уровень передачи данных протокола HTTP является «пассивным наблюдателем», и его роль ограничивается передачей запросов SOAP от клиента серверу с использованием метода POST. Детали сервисного запроса, такие как имя удаленной процедуры и входные аргументы, кодируются в теле запроса. Архитектура REST, напротив, рассматривает уровень передачи данных HTTP как активного участника взаимодействия, используя существующие методы НТТР, такие как GET, POST, PUT и DELETE, для обозначения типа запрашиваемого сервиса. Следовательно, с точки зрения разработчика, запросы REST в общем случае более просты для формулирования и понимания, так как они используют существующие и хорошо понятные интерфейсы HTTP.

 Модель SOAP поддерживает определенную степень интроспекции, позволяя разработчикам сервиса описывать его API в файле формата Web Service Description Language (WSDL, язык описания веб-сервисов). Создавать эти файлы довольно сложно, однако это стоит затраченных усилий, поскольку клиенты SOAP могут автоматически получать из этих файлов подробную информацию об именах и сигнатурах методов, типах входных и выходных данных и возвращаемых значениях. С другой стороны, модель REST избегает сложностей WSDL в угоду более интуитивному интерфейсу, основанному на стандартных методах HTTP, описанных выше.

- В основе REST лежит концепция ресурсов, в то время как SOAP использует интерфейсы, основанные на объектах и методах. Интерфейс SOAP может содержать практически неограниченное количество методов; интерфейс REST, напротив, ограничен четырьмя возможными операциями, соответствующими четырем методам HTTP.

**REST лучше использовать и почему**:

1. В сервисах, которые *будут использоваться из javascript*. Тут и говорить нечего, javascript хорошо работает с json, поэтому именно его и надо предоставлять.
2. В сервисах, которые будут использоваться из языков, в которых нет возможности сгенерировать прокси клиента. Это *Objective-C*, например. Не нужно парсить вручную SOAP-конверт, это незачем.
3. Когда существуют *очень высокие требования к производительности*. Это, как правило, очень интенсивно используемые API, вроде Twitter API или Google API.

**А когда же использовать SOAP:**

1. Когда взаимодействие происходит между платформами, под которые *существуют инструменты для ускорения разработки с использованием SOAP*. Например, правильно писать SOAP-сервис на JAVA, который будет использоваться из .Net и Flex.
2. Когда можно извлечь пользу из всех накруток SOAP. Достоверных сведений о том, что кто-то сумел это сделать, у меня нет. Понятный пример придумать не могу.

# Required Automation Instrument

### Aspect Oriented Programing

### Fluent Interface

### .Net Reports to MTM

### Reflection Mechanism

# Automation test runs and CI integration

## Automated build concept

Continuous integration (CI) is the practice, in software engineering, of merging all developer working copies with a shared mainline several times a day.Later elaborations of the concept introduced build servers, which automatically run the unit tests periodically or even after every commit and report the results to the developers.

<http://en.wikipedia.org/wiki/Continuous_integration>

## Integrating CI and version control

In Jenkins this type of integration is made using various plugins.

## Building test scripts with the CI process

Test scripts can be divided on some categories. Some smote tests be ran during after commit build to make sure that all basic functionality is working fine.

## Executing CI scripts

There are two ways to run CI scripts. First one is to trigger build manually and second is to configure some riggers to start build automatically.

## Monitoring build process

In Jenkins it is possible to monitor build process with help of console output functions.

## Build status reporting (notification)

Jenkins have a lot of various plugins for informing about build status. Also there is a custom possibility to send e-mails after build actions are complete.

## Configuring user’s authorization

You can implement user authorization on project level and site level. For detailed information visit Jenkins wiki.

<https://wiki.jenkins-ci.org/display/JENKINS/Authentication>

## Configuration reports publication and auto-email notifications

Also can be done in post-build section with help of different plugins.

## How to configure and launch UI tests on remote slave via CI

First it is important to start slave and connect it to master. After connection is successfully established it is possible to configure Jenkins to run some jobs on this slave.

# Performance Testing

## Load Testing Concept

Load Testing includes increasing the Load and see how the system behaves under higher load. During Load Tests you can monitor response times, throughput, server condition and much more. The goal of Load Testing is not to break the target environment though.

Goal: Find metrics for system performance under high load

<http://www.soapui.org/load-testing/concept.html>

## JMeter for Load Testing

### Scenario recording

There are two options for creating test scripts for Jmeter. U can manual create HTTP requests or use HTTP Test Script Recorder.

### Parallel test execution

If you require to run more than one scenario in Jmeter, and want the scenarios to run at the same time, you can simply add another Thread Group to your test plan. In your test plan you can configure both thread groups to start at the same time.

There is no guaranty that the requests will be in the exact same time, but both Thread Groups will start simultaneously.

### Analyzing logs

JMeter can be configured to save the results in different formats. I can be XML format as well as CSV format. It can be easily parsed and presented as a graph.

The attributes are just shortened and have the following meaning

by: Bytes

de: Data encoding

dt: Data type

lb: Label

lt: Latency = time to initial response (milliseconds) - not all samplers support this

rc: Response Code (e.g. 200)

rm: Response Message (e.g. OK)

s: Success flag (true/false)

t: Elapsed time (milliseconds)

tn: Thread Name

ts: timeStamp (milliseconds since midnight Jan 1, 1970 UTC)

9 TIPS ON SMART PERFORMANCE TEST REPORTING  
Your key to smart reporting as part of quality software testing is presenting the information to the audience in a simple, quick and intuitive manner.  
Here are 9 underlying principles to follow for effective reporting of your performance testing results.  
1) Report Early and Often  
Constant sharing of data and information is crucial to get the overall success of your performance testing project. For an effective approach to this, send stakeholders the summary tables and charts every couple of days in an email which contains concise statements for all key points.  
2) Use Visuals  
Many people find statistics and data reported a visual format easier to digest. Especially this is true with performance results data in which the data volume is often large and most valuable findings result from identifying patterns in your data.  
3) Make Your Report Intuitive  
The issues of your report should be intuitively and quickly clear from your presentation. A method to ensure the intuitiveness is removing all identifiers and labels from your charts and graphs as well as all the identifying information from the narratives.  
4) Use Appropriate Statistics  
Despite a widespread need for understanding multiple statistical concepts, there are software testers, developers and managers who are bad at it. Therefore, if you don’t have confidence in what statistics you should use in order to cove a certain issue, ask for assistance.  
5) Consolidate Your Data Correctly  
It’s not strongly necessary to consolidate your results, but in this way demonstrating patterns of the results consolidated into a couple of graphs tends to be easier. Additionally, remember that only identical and statistically similar test executions’ results may be consolidated into your performance report’s charts and tables.  
6) Summarize Your Data Effectively  
When you summarize the results, they are more likely to better demonstrate their meaningful patterns. Summary your charts and tables to present the data from various test executions alongside so that patterns and trends can be easily identified.  
7) Customize Your Reports  
There are three groups of people who usually read your performance testing reports: technical, non-technical members of your team and stakeholders who aren’t in the team. While reporting, ensure you know your audience and its expectations before you decide in what way you should present your results.  
8) Make Concise Verbal Summaries  
Your results should contain at least a brief verbal summary while there are also results which are most easily to present in writing. Decide on what to include in such summaries based on the intended audience.  
9) Make Your Data Available  
Your data provides different people with different value at different times. By making it available you’ll also minimize the common perception that your performance results are just fabrications based on some set of processes and tools they don’t understand.  
Use the above principles every time you’re set to deliver a great report of performance test results your quality software testing strategy will draw on.

## Capacity Testing

To determine how many users and/or transactions a given system will support and still meet performance goals.

Capacity testing is conducted in conjunction with capacity planning, which you use to plan for future growth, such as an increased user base or increased volume of data. For example, to accommodate future loads, you need to know how many additional resources (such as processor capacity, memory usage, disk capacity, or network bandwidth) are necessary to support future usage levels.

Capacity testing helps you to identify a scaling strategy in order to determine whether you should scale up or scale out.

<https://msdn.microsoft.com/en-us/library/bb924357.aspx>

## Endurance Testing

Endurance testing involves testing a system with a significant load extended over a significant period of time, to discover how the system behaves under sustained use. For example, in software testing, a system may behave exactly as expected when tested for 1 hour but when the same system is tested for 3 hours, problems such as memory leaks cause the system to fail or behave randomly.

<http://istqbexamcertification.com/what-is-endurance-testing-in-software/>

## Web UI performance testing

Some trace log can be added to regular UI tests to measure the performance of UI components.

## Planning performance testing

### Identifying scenarios

* Criticality – How critical is the scenario to the overall operation of the application?
* Frequency – How often is the scenario expected to be performed?
* Difficulty – How difficult is the scenario to simulate?
* Verifiability – Can correct operation of the scenario be easily verified?

<http://www.webperformance.com/load-testing/blog/2010/09/choosing-the-scenarios-for-a-load-test/>

### Parameterizing tests

Parameterization refers to the ability of the tester to change various parameters in a load test on a just-in-time basis. The simplest example would be a test where each virtual user logs in to the tested application with a different UserID and Password. These UserIDs and passwords might be retrieved from a text file, a database table or they might even be auto generated by the load test software.

There are a number of reasons. Here are a few examples:

* Your application might not allow a user to be logged on more than once simultaneously.
* Through the use of different parameter values, you can avoid database caching.
* You might want to bring back multiple types of response data, and the type of response data might be a direct result of a parameter in the request.
* Requests to a database table will do a much better job of testing your indexes if they are calling for, or are inserting, a wide variety of data.
* It is more realistic for users to behave as individuals rather than as clones of each other.
* The ability to parameterize your URLs makes it easy for you to run the same test against different environments. Rather than re-scripting the test, you can merely change one parameter value.

<http://blog.smartbear.com/performance/load-testing-101-parameterizing-performance-tests/>

### Metrics

Usually time, RAM usage, processor time, request transfer speed are measured while doing performance test.

## Performance tests reporting

### SUT metrics collection

The system under test can send realtime data as follows to provide good visibility. This could be anything from cpu utilisation to database reads.

### Test environment metrics collection

Usually it is recommended to collect the following environment metrics: Processor, RAM, HDD, OS.

# Web profile requirements

## Typical Web UI Automation Tasks

### WebDriver concept

Selenium is a portable software testing framework for web applications. Selenium provides a record/playback tool for authoring tests without learning a test scripting language (Selenium IDE). It also provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including Java, C#, Groovy, Perl, PHP, Python and Ruby. The tests can then be run against most modern web browsers. Selenium deploys on Windows, Linux, and Macintosh platforms. It is open-source software, released under the Apache 2.0 license, and can be downloaded and used without charge.

<http://en.wikipedia.org/wiki/Selenium_(software)>

### Auto-generated IDs

There are few ways to handle the problem of auto-generated IDs

* Absolute XPath
* Closest stable element
* Stable preceding text
* Text content

<http://www.optimusinfo.com/dynamic-ids-headache-automation-great-challenge/>

### Multiple browser support

Selenium supports multiple browser testing. For using cross-browser feature you have to install all required additional software and instantiate driver of exact browser type that is needed.

### xPath selectors

XPath, the XML Path Language, is a query language for selecting nodes from an XML document.

PROS:

Allows very precise locators

CONS:

Slower than CSS

Relies on browser’s XPath implementation which is not always complete (especially on IE) and as such is not recommended for cross-browser testing

<http://hedleyproctor.com/2011/05/tutorial-writing-xpath-selectors-for-selenium-tests/>

<http://www.silverwareconsulting.com/index.cfm/2013/3/8/Very-nice-xpath-and-css-selector-locator-cheat-sheet-for-Selenium>

### css Selectors

CSS selectors allow you to select and manipulate HTML elements. CSS selectors are used to "find" (or select) HTML elements based on their id, class, type, attribute, and more.

PROS:

Much faster than XPath

Widely used

Provides a good balance between structure and attributes

Allows for selection of elements by their surrounding context

CONS:

They tend to be more complex and require a steeper learning curve

<http://seleniumeasy.com/selenium-tutorials/css-selectors-tutorial-for-selenium-with-examples>

### ID selectors

The Id strategy looks for an element in the page having an id attribute corresponding to the specified pattern. <label id="my\_id" /> will be matched by a locator like id=my\_id or just my\_id

PROS:

Each id is supposed to be unique so no chance of matching several elements

CONS:

Works well only on elements with fixed ids and not generated ones

<https://newcircle.com/static/bookshelf/selenium_tutorial/locators.html>

### Handling Ajax Requests

It is recommended to use WebDriverWait class for handling Ajax requests.

<http://blog.devscrum.net/2011/04/selenium-2-webdriver-handling-ajax-page-requests/>

### JavaScript execution using WebDriver interface

You can execute arbitrary javascript to find an element and as long as you return a DOM Element, it will be automatically converted to a WebElement object.

### Load testing Web UI

To perform load testing it is possible to run selenium tests in different threads. To spread the load of testing across several machines you can use Selenium Grid.

### Measuring performance of the user interface

There are a lot of factors that you should think over when measuring performance. First of all network latency.   
Thou it is possible to add some counters to page object methods to measure UI performance. Another good idea is to write those values to some file in order to provide some statistics.

<http://ux.stackexchange.com/questions/21951/how-to-measure-ui-lag-on-computer>

### TCP vs UDP

Сначала мы должны запустить центарльный сервер (hub). Это делается с помощью следующей команды:

java -jar selenium-server-standalone-2.11.0.jar -role hub

После запуска команды можно перейти на панель администрирования хаба по адресу:

http://localhost:4444/grid/

Запускаем узлы кластера:

java -jar selenium-server-standalone-2.11.0.jar -role webdriver -hub http://localhost:4444/grid/register -port 5556

Е