Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

**Ітеративний та інкрементний методи розробки програмного забезпечення**

**Текстова частина до курсової роботи**

**за спеціальністю „Прикладна математика” 6.050103**

Керівник курсової роботи к.т.н., доц. Афонін А.О.

*\_\_\_\_\_\_\_\_\_\_\_\_(підпис)*

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2014 р.

Виконав студент

Каплунський Я.О.

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2014 р.

Київ 2014

**Зміст**

Вступ……………………………………………………………….…....…………3

**Розділ 1: Розробка програмного забезпечення................................................4**

1.1 Парадигма програмного забезпечення..................................................4

1.2Waterfall...................................................................................................6

1.3 Ітеративна та інкрементна розробка......................................................9

**Розділ 2: Інструменти розробки........................................................................12**

2.1 TDD.........................................................................................................12

2.2 Refactoring..............................................................................................13

2.3 Kanban....................................................................................................14

2.4 Agile........................................................................................................16

2.5 Scrum......................................................................................................18

**Розділ 3: Демонстраційний проект...................................................................20**

3.1 Image Tool..............................................................................................20

3.2 Архітектура............................................................................................20

3.2.1 MVC...........................................................................................20

3.2.2 Factory.......................................................................................22

3.2.3 Singleton....................................................................................22

3.2.4 Strategy......................................................................................23

3.2.5 Chain of responsibility...............................................................24

Висновок.................................................................................................................25

Список використаної літератури..........................................................................26

**Вступ**

Розробка програмного забезпечення це не тільки програмування складних систем це також ведення документації та тестування програмного продукту, включаючи в себе налаштування взаємозв’язків між різними компонентами програми. Таким чином розробка ПЗ може включати в себе: дослідження, нові розробки, створення прототипів, зміни уже існуючого продукту, а токож будь-які інші методи які в результаті дадуть програмний продукт.

Метою роботи є дослідження різних методів розробки програмного забезпечення та застосування ітеративного та інкрементного методів для реалізації шаблонізатора зображень, використовуючи open-source бібліотеку ImageMagick.

* Телекомунікаційній компанії потрібно було змінити 3 тисячі стрічок в базовому ПЗ об’єм якого приблизно 1 000 000 строк. Час, який потрібний був, включаючи аналіз потреб, планування, реалізацію і тестування, зайняло 9 годин[1].
* Група розробників ПЗ для BBC США почала розробку проекта з рік з бюджетом в $2 000 000, хоча інші компанії пропонували термін 2 роки та бюджет до $100 000 000. Коли ця група розробників здала проект на місяць швидше запланованого терміну, менеджер проекту сказав, що успіх був досягнутий за рахунок методик, які відомі уже дуже давно, але чомусь рідко використовуються[1].

Розробка програмно продукту приносять мільярди доларів в світову економіку тому методології розробки стають вкрай важливі.

# **1. Розробка програмного забезпечення**

## 1.1 Парадигма програмного забезпечення

“*The only way to go fast is to go right*”

*(Uncle Bob)*

Для всіх буде не секрет, якщо я скажу що IT - індустрія проникла, буквально, в кожну сферу людського життя. Ми дізнаємось погоду з інтернету, рухаємось по маршруту який на проклав навігатор. Проблема, яка залишилася для розробників програмного забезпечення одна – ніхто не знає як писати якісний програмний продукт. Ми постійно спостерігаємо як найкращі ІТ компанії світу допускають помилки у своїх продуктах. Перед компаніями постійно постають три фундаменталні проблеми:

* Робити правильні речі
* Робити речі правильно
* Робити швидко

Робити правильні речі, що це може значити? Це значить, що потрібно робити продукт який комусь потрібний, продукт який взмозі вирішувати конкретні задачі.

Робити правильно – напевно очевидно, що програмний продукт повинен бути якісним, користувач не повинен знати що таке аварійне завершення програми, данні користувача не можуть бути втрачені.

Третє проблема – швидкість. Користувачів, та замовників не цікавить як буде зроблена програма їх цікавить коли. Швидка розробка ПЗ також дає і перевагу програмістам, оскільки, вони роблять програму швидко - у них є можливість спробувати різні технології, швидко реагувати на зміну ринку.

Так на приклад гру *Duke Nukem Forever* почли розробляти в далекому 1996, а реліз випустили в 2011. Здавалось, що за такий час можна створити «ідеалну гру», але вона не виправдала надій , які на неї покладали[2].

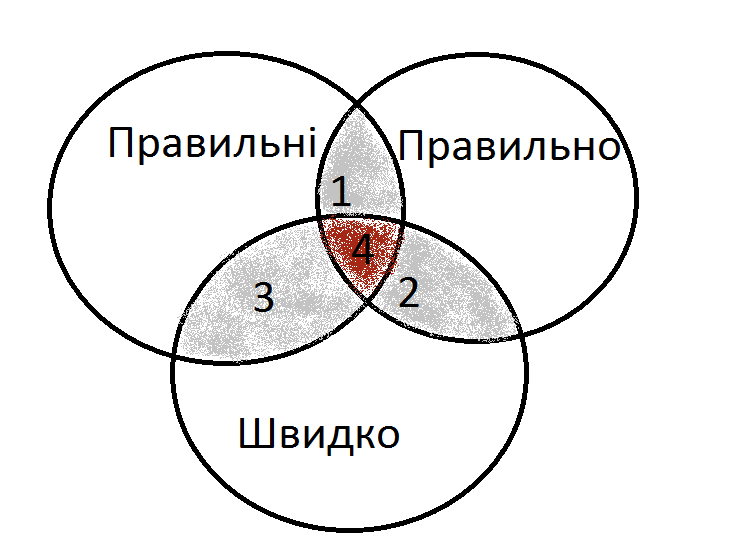


Рис.1.1.1

1 – бачимо, можливість втратити ринкову інертність

2 – випускається програмний продукт який нікому не потрібний

3 – продукт який потрібний клієнтам, але який чекають великі проблеми в майбутньому при модернізації та підтримці.

4- «золота середина»

Тому для досягнення поставлених цілей почали використовувати різні методології розробки ПЗ. Перші згадки про використання методологій до розробки ПЗ датуються кінцем 60 – х років XX століття[3]. Тоді для розробки програмного забезпечення взяли на озброєння Waterfall methodology – яку використовували при розробці літаків[4]. З того часу багато методологій пережили свої злети та падіння. Новими в цьому є : cleanroom, XP, incremental, iterative, V- model, spiral.

## 1.2 Waterfall

Водоспадна модель – послідовний метод розробки програмного забезпечення, через що і отримала таку назву. Модель використовували за довго до розробки програмного забезпечення, вона використовувалася при конструюванні мостів, літаків та інших складних інженерних конструкцій.

Простий вигляд цієї моделі:

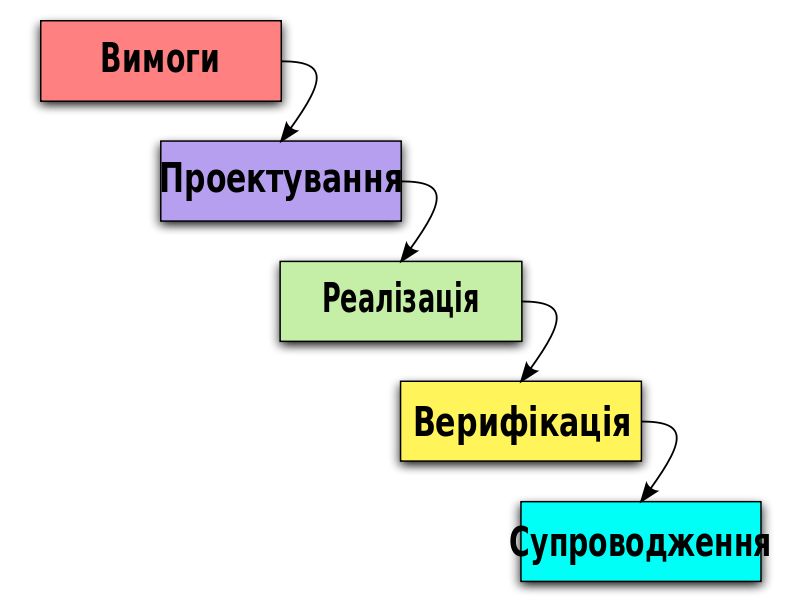


Рис. 1.2.1

Для данного підходу характерний перфекціонізм на кожному кроці, повернення до виправлення помилки на попередні кроки тягнутиме великі ресурсні затрати.

Перший формальний опис моделі для використання при розробці програмного забезпечення дав В.В. Ройс у 1970 році[4]. Він проаналізував проблеми та способи застосування цієї методики при розробці програмного забезпечення .

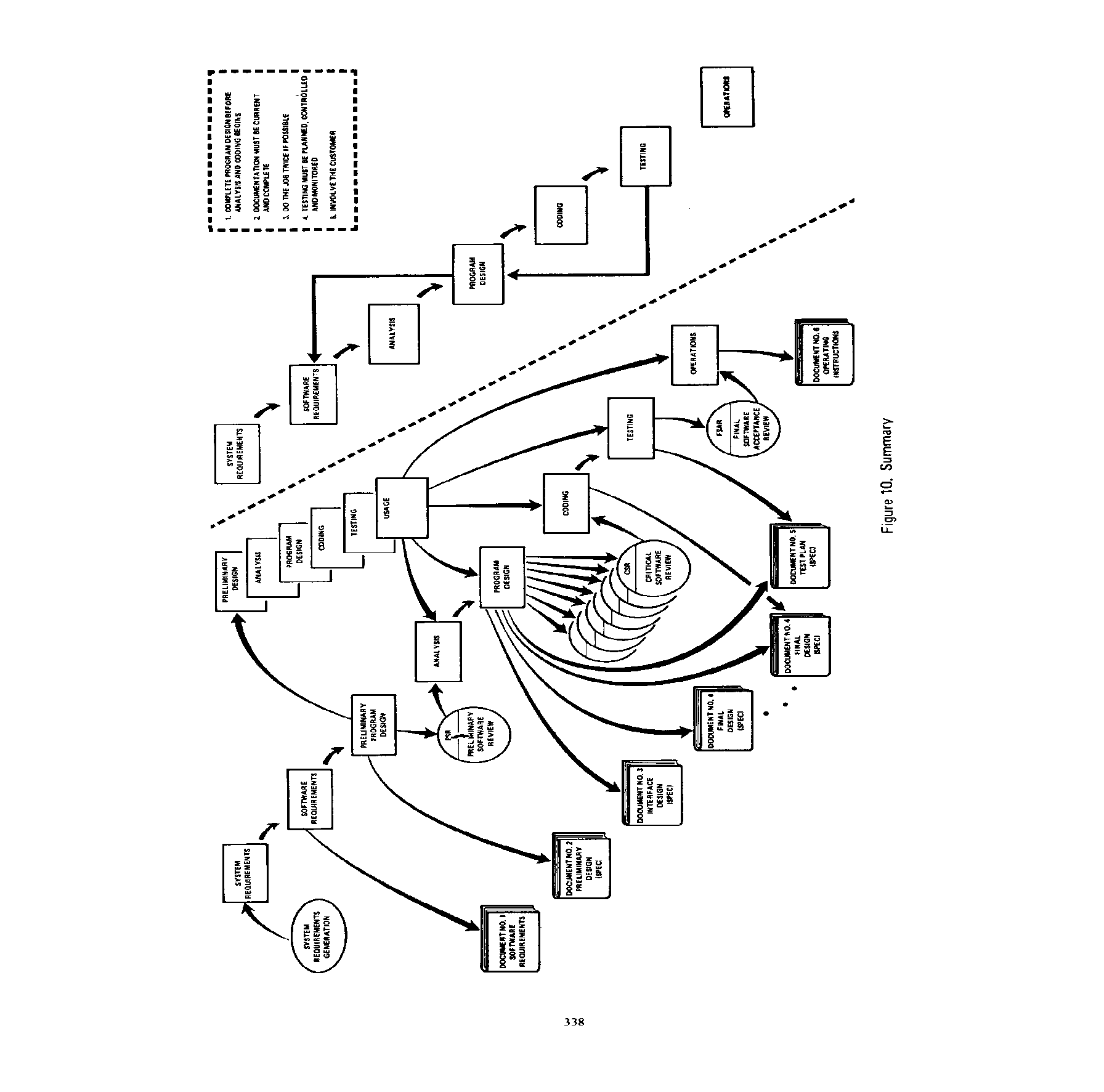


Рис. 1.2.2

Діаграма (рис. 1.2.2) , за словами Ройса переводить ризиковану розробку програмного забезпечення, в розробку продукта, який потрібний замовнику[4]. Хотілось би зазначити, що кожен крок в продемонстрованому підході коштує трішки більше. Зрозуміло, що простійший опис методу, зменшою кількістю кроків також буде працювати вдало і ці додаткові кошти були б використанні не правильно, але коли мова йде про розробку великих проектів, то як показує практика – прості методи ніколи не спрацьовують, тому додаткове фінансування на кожному кроці просто необхідне[4]. Доречно також підмітити те, що Ройс при описі цієї моделі, передбачив ітераційний процес, хоча сама модель є лінійною.

## 1.3 Ітеративна та інкрементна розробка

Ітеративна та інкриментна розробка - це серцевина циклічного процесу розробки програмногозабезпечення, який був розроблений у відповідь на слабкі сторони водоспадної моделі. Процес починається з початкового планування і закінчується впровадженням з циклічними взаємодіями між цими етапами.[9]



Рис.1.3.1

Основна ідея полягає в тому, щоб розробити систему шляхом циклів, що повторюються (ітеративний) та в менші проміжки часу (інкрементний), даючи змогу розробнику скористатися перевагами того, що було вивчене під-час розробки попередніх порцій або версій системи. Навчання проходить під час реалізації програми, з кожною ітерацією, заглиблюючись глибше. На кожному кроці(ітерації) виконуються зміни дизайну та додаються нові функціональні можливості.

Процедура складається з кроку ініціалізації, ітеративного кроку та контрольного списку проекту. Крок ініціалізації створює базову версію системи. Метою початкової реалізації є створення продукту на який можуть відгукнутися користувачі. Цей продукт повинен запропонувати набір зразків ключових аспектів проблеми і забезпечити розв'язання, яке буде достатньо простим для того, щоб легко зрозуміти і реалізувати його. Для керування ітеративним процесом створюється контрольний список проекту, який вміщує записи усіх завдань, які потрібно виконати. Він включає такі елементи: нові можливості програми, які потрібно реалізувати та частини поточного рішення, які потрібно переробити. Контрольний список постійно змінюється, в результаті фази аналізу.

Ітерація включає реконструювання та реалізацію завдань з контрольного списку, а також аналіз поточної версії системи. Рівень деталізації проектного плану не диктується ітеративним підходом. У невеликих ітеративних проектах код може представляти джерело документації системи. Проте, у критичних до організації проектах може використовуватися формальний документ-план програмного продукту. Аналіз ітерації базується на відгуках користувачів та на доступних засобах аналізу програми. Він включає аналіз структури, модульності, зручності у використанні, надійності, продуктивності та досягнення поставлених цілей. Контрольний список проекту змінюють на основі результатів аналізу.



Рис.1.3.2

# **2 Інструменти розробки**

## 2.1 TDD

Test-driven development – [технологія розробки програмного забезпечення](http://uk.wikipedia.org/w/index.php?title=%D0%A2%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%8F_%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&action=edit&redlink=1), яка використовує короткі ітерації розробки, що починаються з попереднього написання тестів, які визначають необхідні покращення або нові функції. Кожна ітерація має на меті розробити код, який пройде ці тести.

Один із ключових моментів TDD полягає у тому, що підготовка тестів перед написанням самого коду пришвидшує процес внесення змін. Варто зауважити, що розробка через тестування є методологією розробки програмного забезпечення, а не його [тестування](http://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)[11].

В TDD підході програмісти використовують *Red-Yellow-Green* підхід, який полягає в наступному:

* Red – написання тесту, що буде покривати проблемну ділянку коду.
* Yellow – після того, як тест провалився зробити уся для того, щоб він пройшов вдало
* Green – зробити рефакторинг написаному коду

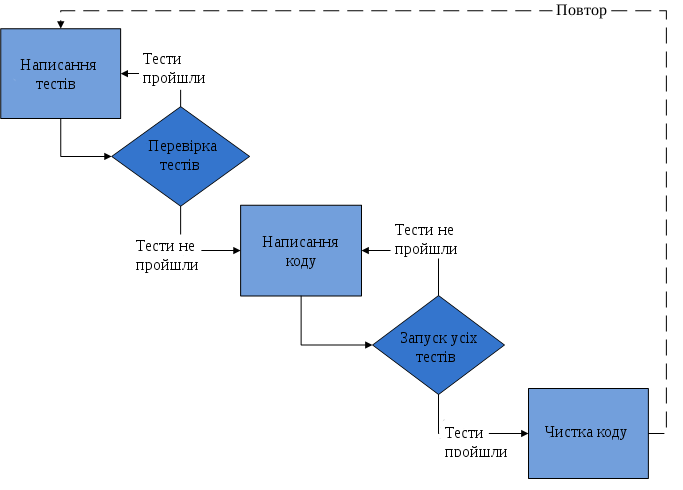


Рис.2.1.1

При розробці тестових кейсів досить часто використовують використовують бібліотеки для тестування, створення та автоматизації запуску наборів тестів. На практиці, в добре написаній програмі, модульні тести повинні покриваюти не менще 90% ділянок коду.

## 2.2 Refactoring

Рефакторинг тісно пов'язанаий з такими принципами як (*keep it simple, stupid,*[*KISS*](http://uk.wikipedia.org/wiki/KISS_(%D0%BF%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF))), (*you ain't gonna need it,*[*YAGNI*](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_YAGNI)). Кент Бек також пропонує принцип (*fake it till you make it*). Роберт Мартін говорить про проблему *(don't repeat yourself DRY).* Головним завданням рефакторингу є перетворення [програмного коду](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4), зміна внутрішньої структури [програмного забезпечення](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F) для полегшення розуміння коду і легшого внесення подальших змін без зміни зовнішньої поведінки самої системи.

Слово «рефакторинг» пішло від терміу [«факторинг»](http://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3) в [структурному програмуванні](http://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%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), який означав [декомпозицію](http://uk.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%B7%D0%B8%D1%86%D1%96%D1%8F) програми на максимально автономні та елементарні частини.[7]

1. Прийоми, що дозволяють розбити код на дрібніші, зрозуміліші частини.
   * Відокремлення методу (Extract Method).
   * Відокремлення базового класу (Extract Superclass).
2. Прийоми, що дозволяють забезпечити додаткову [абстракцію](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)).
   * [Інкапсуляція](http://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) поля (Encapsulate Field) — замінює прямий доступ до поля на доступ через методи-аксесори.
   * Узагальнення типу (Generalize Type) — заміна типів, з якими працює клас, на більш узагальнені.
   * Заміна блоків перевірки типів на шаблони [«Стан» (State)](http://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) або [«Стратегія» (Strategy)](http://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D1%96%D1%8F_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)).
   * Заміна умовних операторів [поліморфізмом](http://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%96%D0%BC%D0%BE%D1%80%D1%84%D1%96%D0%B7%D0%BC_(%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)" \o "Поліморфізм (програмування)).
   * Створення поля або локальної [змінної](http://uk.wikipedia.org/wiki/%D0%97%D0%BC%D1%96%D0%BD%D0%BD%D0%B0_(%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)" \o "Змінна (програмування)) (Introduce Field/Introduce Local Variable).
3. Прийоми, що змінюють назви членів та їх розташування.
   * Переміщення методу/поля (Move Method/Field) в інші класи або файли коду.
   * Перейменування члена (Rename) — зміна імені, з автоматичною заміною всіх посилань на старе ім’я в коді.
   * Переміщення члену до базового/дочірнього класу (Pull Up/Push Down)[11]

## 2.3 Kanban

Це система планування для відкладеного та точно спланованого (*just-in-time JIT*) виробництва. Kanban –це система контролю логічного ланцюжку з точки зору виробництва, а не управління ресурсами. Вперше цей підхід був застосований на заводі Toyota в 1953 році, для поліпшення та підтримки високого рівня виробництва. Сьогодні досить часто прирозробці програмного забезпечення користуються цим підходом. Основна ідея цього підходу полягає в тому, що за процесом виробництва можна спостерігати *just-in-time.* Для цього спеціально створюється дошка на якій позначають на якій стаії розробки знаходиться той чи інший процес.

Trello, JIRA, Target Process, issues tracking на https://bitbucket.org/ та https://github.com/ та багато програмних продуктів, які допомагають створити таку дошку. Я використовував ресурс trello. Робота в ньому виглядала так:

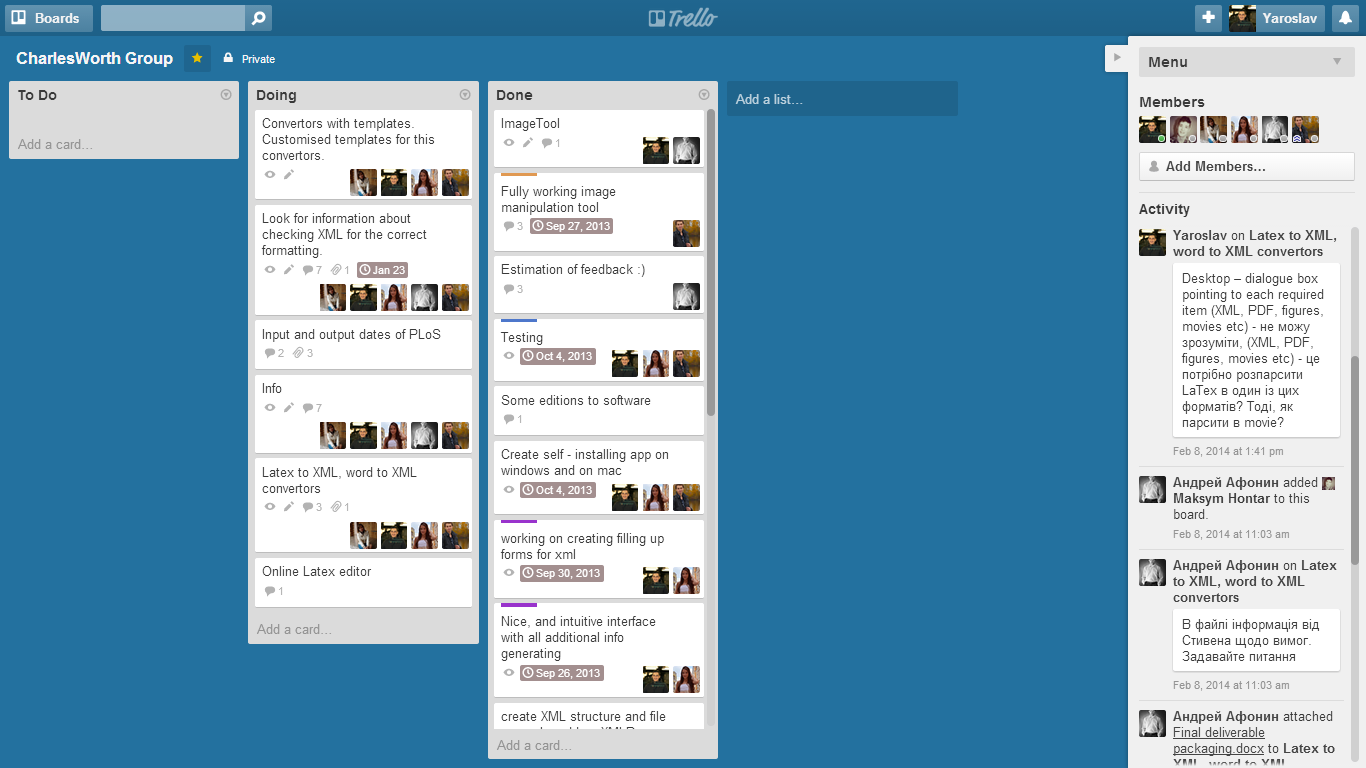


Рис.2.3.1

На цьому рисунку можна побачити, як за допомогою цієї системи можна легко організувати роботу команди.

## 2.4 Agile

Гнучкаа розробка програмного забезпечення — клас методологій розробки програмного забезпечення, що базується на [ітеративній розробці](http://uk.wikipedia.org/wiki/%D0%86%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0_%D1%82%D0%B0_%D1%96%D0%BD%D0%BA%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D0%BD%D0%B0_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0), в якій вимоги та розв'язки еволюціонують через співпрацю між самоорганізовуваними багатофункціональними командами. Гнучка розробка - найкращий засіб для підвищення продуктивності розробників програмного забезпечення.

Більшість гнучких методологій націлені на мінімізацію ризиків, шляхом зведення розробки до серії коротких циклів, що мають назву [ітерацій](http://uk.wikipedia.org/wiki/%D0%86%D1%82%D0%B5%D1%80%D0%B0%D1%86%D1%96%D1%8F), які зазвичай тривають один-два тижні. Кожна ітерація сама по собі виглядає як програмний проект в мініатюрі, і включає всі завдання, необхідні для видачі мінімального приросту за функціональністю: планування, аналіз вимог, проектування, кодування, тестування і документування. Хоча окрема ітерація, як правило, недостатня для випуску нової версії продукту, мається на увазі те, що гнучкий програмний проект готовий до випуску наприкінці кожної ітерації. Після закінчення кожної ітерації, команда виконує переоцінку пріоритетів розробки.

Основною метрикою agile методів є робочий продукт. Віддаючи перевагу безпосередньому спілкуванню agile методи зменшують обсяг письмової документації в порівнянні з іншими методами. Це привело до критики цих методів як не дисциплінованих.

Agile не включає практик, а визначає цінності та принципи, які записані в Agile Manifesto, який розроблений і прийнятий 11-13 лютого 2001. Agile Manifesto містить 4 основні ідеї та 12 принципів.

Основні ідеї:

* Особистості та їхні взаємодії важливіше, ніж процеси та інструменти;
* Робоче програмне забезпечення важливіше, ніж повна документація;
* Співпраця із замовником важливіша, ніж контрактні зобов'язання;
* Реакція на зміни важливіше, 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.[8]

## 2.5 Scrum

Scrum — це кістяк процесу, який включає набір методів і попередньо визначених ролей. Головні дійові особи — *ScrumMaster*, той хто опікується процесами, веде їх і працює як керівник проекту, *Власник Продукту*, людина, що представляє інтереси кінцевих користувачів та інших зацікавлених в продукті сторін, та *Команду*, яка включає розробників.[7]

Протягом кожного спринту , 15-30 денного періоду (тривалість визначається командою), працівники створюють функціональний ріст програмного забезпечення.

Набір можливостей, які імплементуються кожного спринту, приходять з етапу, що має назву product backlog (документація запитів на виконання робіт), який має найвищу пріоритетність за рівнем вимог до роботи, що повинна бути виконана. Запити на виконання робіт (*backlog items*) що визначені протягом наради з планування спринту (sprint planning meeting) переміщуються в етап спринту.

Протягом цієї наради Власник Продукту інформує про завдання, які він хоче, аби були виконані. Тоді Команда визначає, скільки з бажаного вони можуть зробити, щоб завершити необхідні частини протягом наступного спринту. Протягом спринту команда виконує визначений фіксований список завдань. Впродовж цього періоду ніхто не має права змінювати перелік запитів на виконання робіт, що слід розуміти, як заморожування

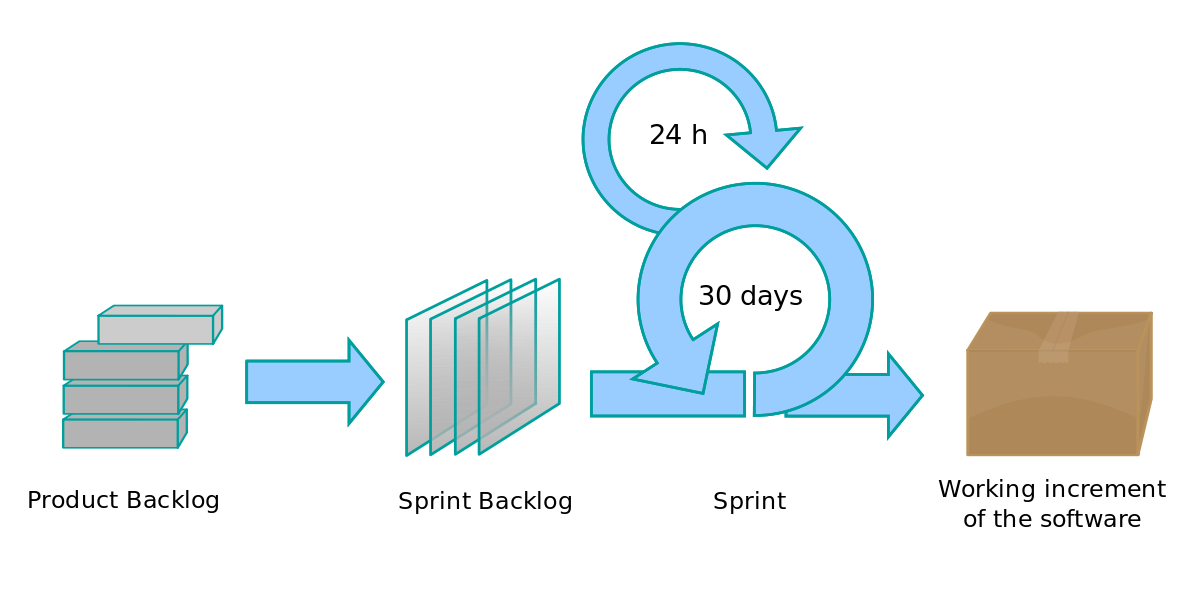
****

Рис.2.5.1

**3 Демонстраційний проект**

## 3.1 Image Tool

Данна програма призначена для шаблонізації зображень, які знаходяться в одній деректорії. Програма реалізована на платформі Java SE з використанням open-source бібліотеки ImageMagick. Інформація про те як потрібно редагувати файли зберігається в XML – файлі, що знаходиться в директорії зображеннями. Для полегшення встановлення на машину, використовується майстер встановлення.

## 3.2 Архітектура

Головна задача програміста – це правильно навчитись керувати складністю, тому в цьому розділі я хочу показати яким чином влаштована архітектура програми, основні її компоненнти та проектні рішення.

При написанні програми я керувався привилом **20/80,** де 20% класів відповідають за 80% функціональності прогами.

## 3.2.1 MVC

(Model-View-Controller) –шаблон який поділяє систему на три частини: модель данних вигляд даних та керування. Застосовується для відокремлення даних (модель) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу.[6]

Головна мета використання цього шаблону – забезпечити гручку архітектуру завдяки послабленню спряжності міжкласами. Логіка відділяється від зовнішнього вигляду програми, а їхні взаємодії здійснюються в окремому компоненті – controller. Такий підхід дозволяє підтримувати продукт, без додаткових зусиль, програма легко може переписуватись.

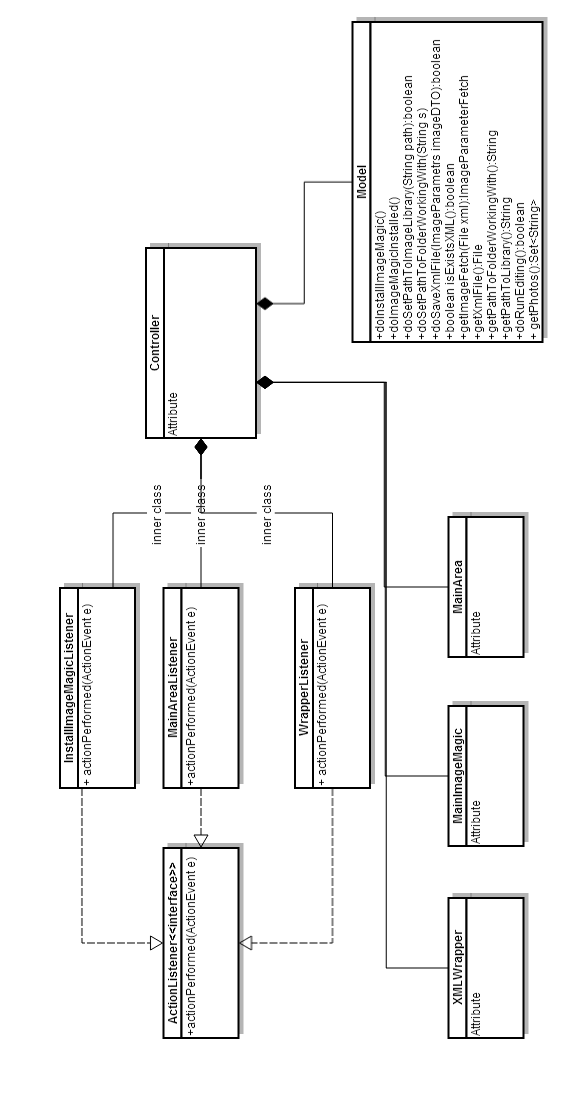


Рис.3.2.1.1

Іншим основним структурним компонентом є фабрика.

**3.2.2 Factory**

Породжуючий шаблон, що визначає інтерфейс для створення об'єкта, але залишає підкласам рішення про те, який саме клас інстанціювати. Фабричний метод, який включається в фабрику, дозволяє класу делегувати інстанціювання підкласам.

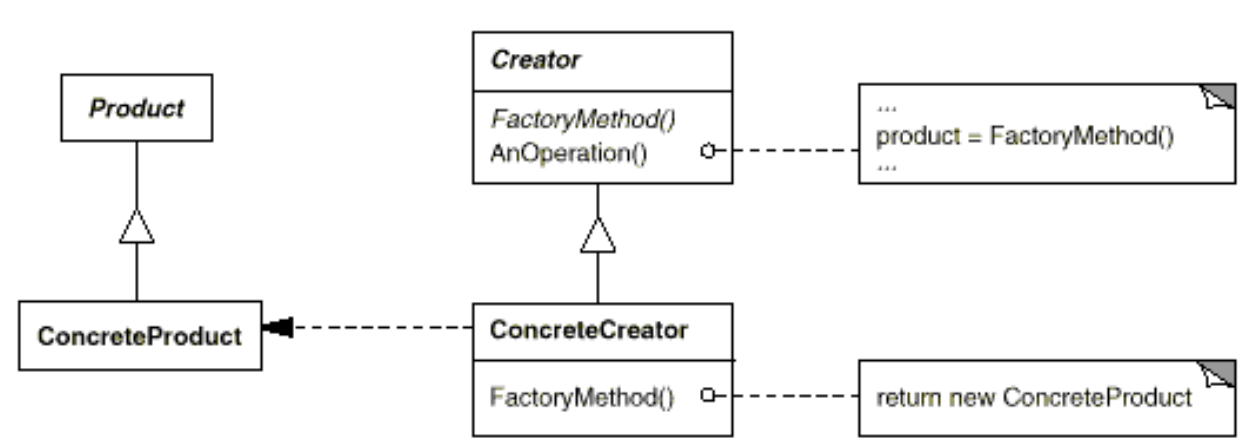


Рис.3.2.2.1

За допомогою фабрики, ми забезпечуємо , однаковий вигляд та поведінку усіх компонентів, які ми використовуємо в прогамі. Найчастіше фабрики використовують у пояднанні з Singleton – ом.

**3.2.3 Singleton**

Це проектний взірець, який гарантує існування не більше одного екземпляру об’єкта в прогамі.

Діаграма паттерну, напевно найпростіша – один клас. Але механізми Java дозволяють нам реалізувати взірень так, як пропонує Joshua Bloch:

public enum Singleton { INSTANCE; }

Ось і все плюси такого підходу очевидні:

* Серіалізація
* Потокобезпечний
* Можливість використовувати EnumSet, EnumMap и т.д.
* Підтримка switch

Також Singleton я використав для організації класу, що відповідає за інформативні повідомлення.

**3.2.4 Strategy**

Цей проектний взірець дозволяє інкаплусювати алгоритми, що дозволяє  міняти вибраний алгоритм незалежно від об'єктів-клієнтів, які його використовують.

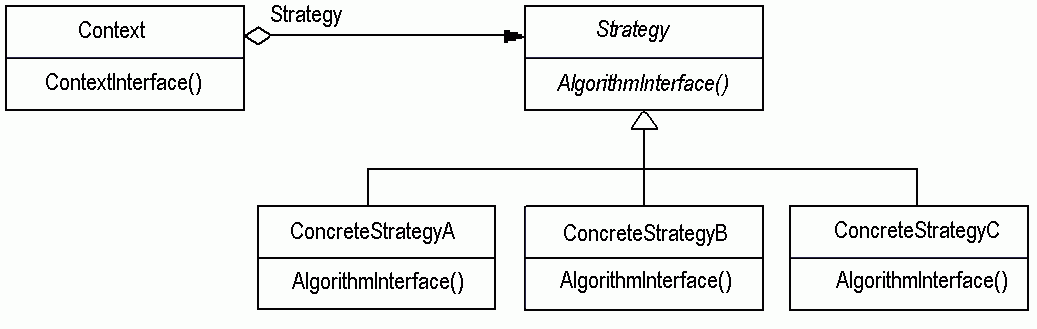


Рис.3.2.4.1

У нашому випадку усі вхідні данні ми зберігаємо в XML-файл, з якого витягуємо інформацію щоразу перед редагуванням зображень. Використання стратегій, вданному випадку, дозволяє полегшити роботу програміста в майбутньому при подальших змінах програмного коду.

**3.2.5 Chain of responsibility**

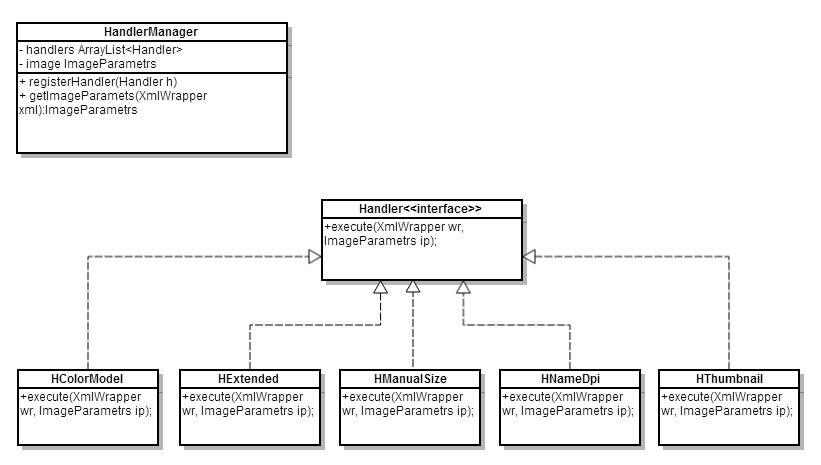
Данний паттерн я застосував для того, щоб можна було легко витягувати інформацію, яку ввів користувач в нашу форму. Використання данного паттерну забезпечує гнучкість програми при доданні нових компонентів. ****

Рис.3.2.5.1

Такий підхід до розробки програмного продукту дозволяє зробти, розробку та підтримку гнучнішою та зручнішою. Данні шаблони є досвідом багатьох програмістів.

**Висновок**

Отже, виконуючи данну роботу було дослідженно два методи розробки програмного забезпечення : ітераційний та інкрементний методи розробки, також за допомогою інструментів цих підходів: refactoring, scrum, Kanban та agile було реалізовано шаблонізатор зображень ImageTool з використанням open-source бібліотеки ImageMagick.

В проекті було реалізовано:

* Редагування зображеннь з різними розширеннями
* Збереження конфігурацій в XML-файлі
* Автозаповнення форми з уже існуючого XML-файлу
* Збереження відредагованих зображень в окремій папці

Не вдалось виправити помилку в ImageMagick:

При переведенні зображення CMYK –модель і з неї в sRGB на зображення накладається ефект сепії.

Програмний код :

Після рефакторингу : <https://bitbucket.org/kapyar/refactorimagemagic>

До : <https://bitbucket.org/kapyar/imagetool>

**Список використаної літератури**

1. Steve McConnell. **Professional Software Development —**  Microsoft Press, **2003**
2. Duke Nukem [Електронний ресурс]. – Режим доступу: URL <http://en.wikipedia.org/wiki/Duke_Nukem_Forever> . **—**  Duke Nukem
3. Geoffrey Elliott **Global Business Information Technology: an integrated systems approach**. **—**  Pearson Education, **2004. — 87 c.**
4. Royce, Winston. **Managing the Development of Large Software Systems / Royce, Winston. —**  TWR*,* **1970. — 328-338 c. —**(The Institute of Electrical And Electronics Engineers)
5. MVC [Електронний ресурс]. – Режим доступу: URL <http://en.wikipedia.org/wiki/Model-view-controller>. **—** MVC
6. Martin Fowler, **Refactoring: Improving the Design of Existing Code/ Martin Fowler —** Addison-Wesley, **1999**
7. Scrum [Електронний ресурс]. – Режим доступу : URL <https://www.scrum.org>
8. Agile manifesto [Електронний ресурс]. – Режим доступу : URL <http://agilemanifesto.org/>
9. Larman, Craig. [**Iterative and Incremental Development: A Brief History**](http://www.craiglarman.com/wiki/downloads/misc/history-of-iterative-larman-and-basili-ieee-computer.pdf)**.** **—** 2003
10. [Erich Gamma](http://en.wikipedia.org/wiki/Erich_Gamma),  [Richard Helm](http://en.wikipedia.org/wiki/Richard_Helm),  [Ralph Johnson](http://en.wikipedia.org/wiki/Ralph_Johnson) , [John Vlissides](http://en.wikipedia.org/wiki/John_Vlissides).[**Design Patterns: Elements of Reusable Object-Oriented Software**](http://en.wikipedia.org/wiki/Design_Patterns_(book)) **—** Addison - Wesley**, 1994**
11. Robert Martin, **Clean Code A Handbook of Agile Software Craftsmanship—** Prentice Hall, **2008**