# ДОДАТОК А

СЛАЙДИ ПРЕЗЕНТАЦІЇ


Рисунок А1 – Слайд презентації № 1

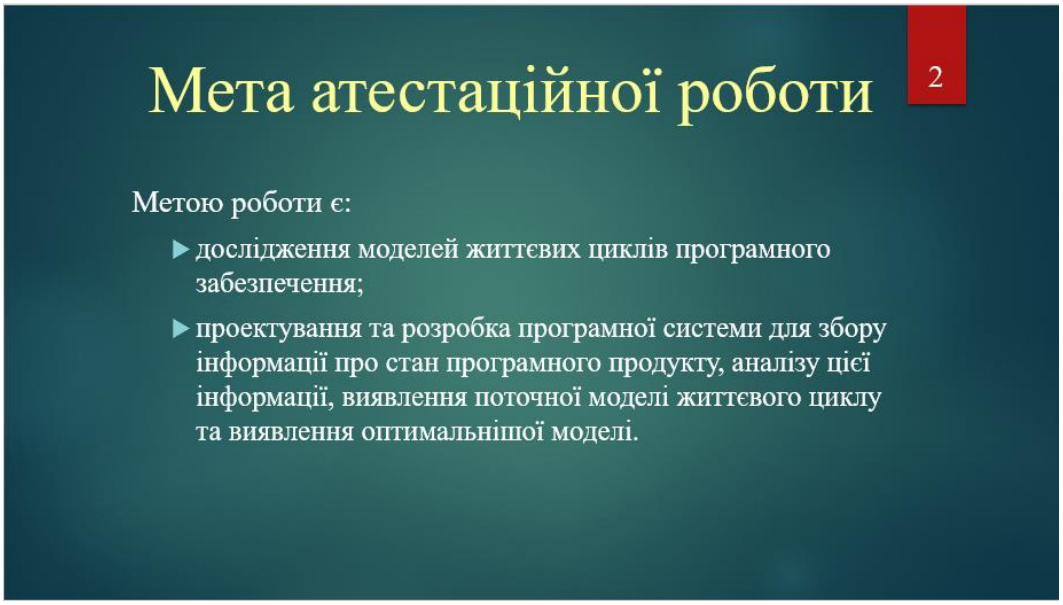


Рисунок А2 – Слайд презентації № 2

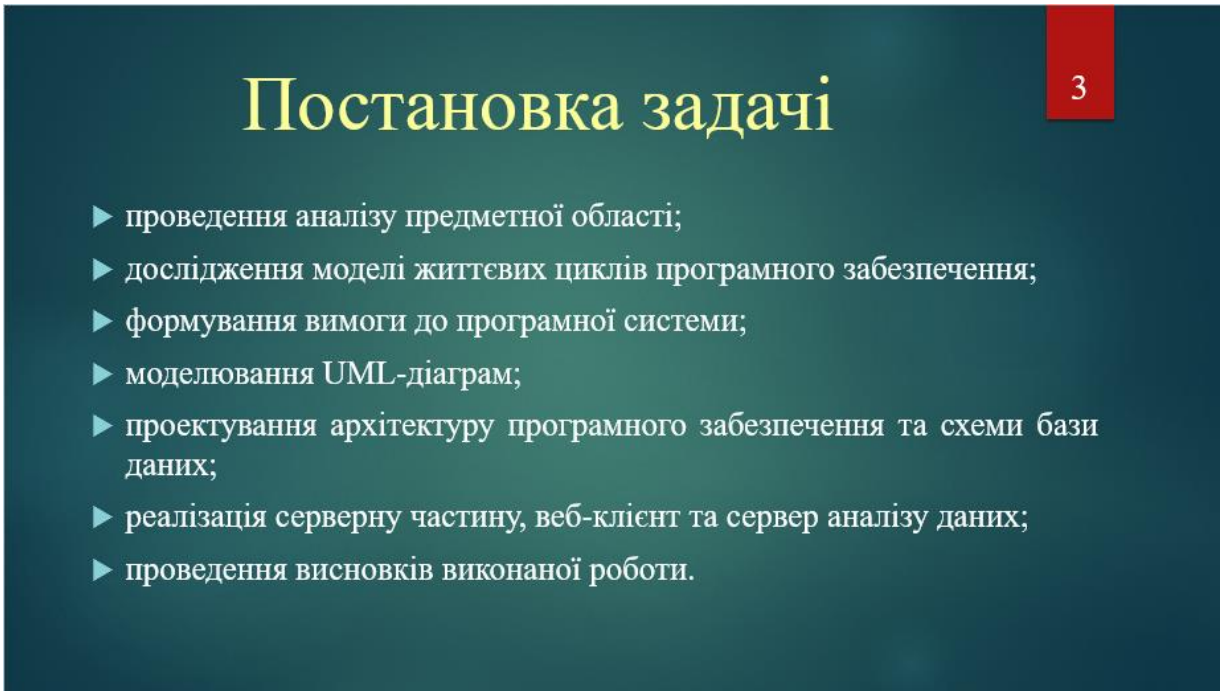


Рисунок А3 – Слайд презентації № 3

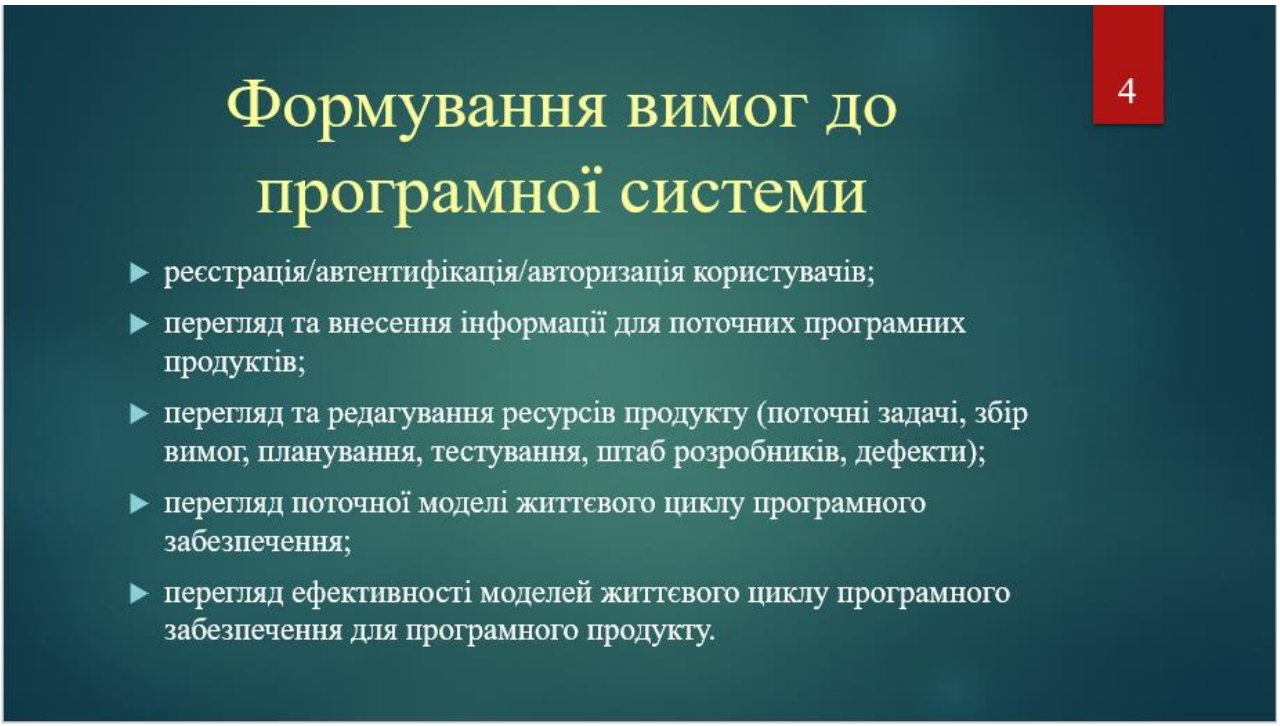


Рисунок А4 – Слайд презентації № 4


Рисунок А5 – Слайд презентації № 5


Рисунок А6 – Слайд презентації № 6


Рисунок А7 – Слайд презентації № 7


Рисунок А8 – Слайд презентації № 8


Рисунок А9 – Слайд презентації № 9

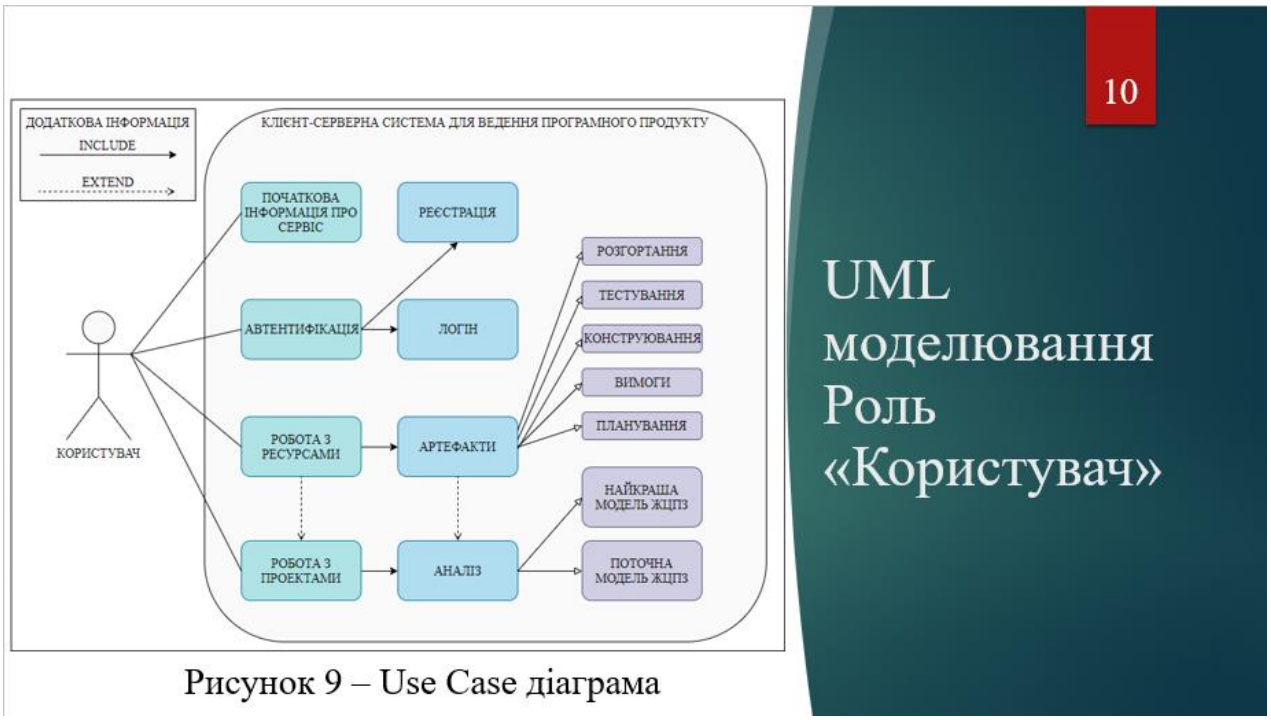


Рисунок 9 – Use Case діаграма

Рисунок А10 – Слайд презентації № 10

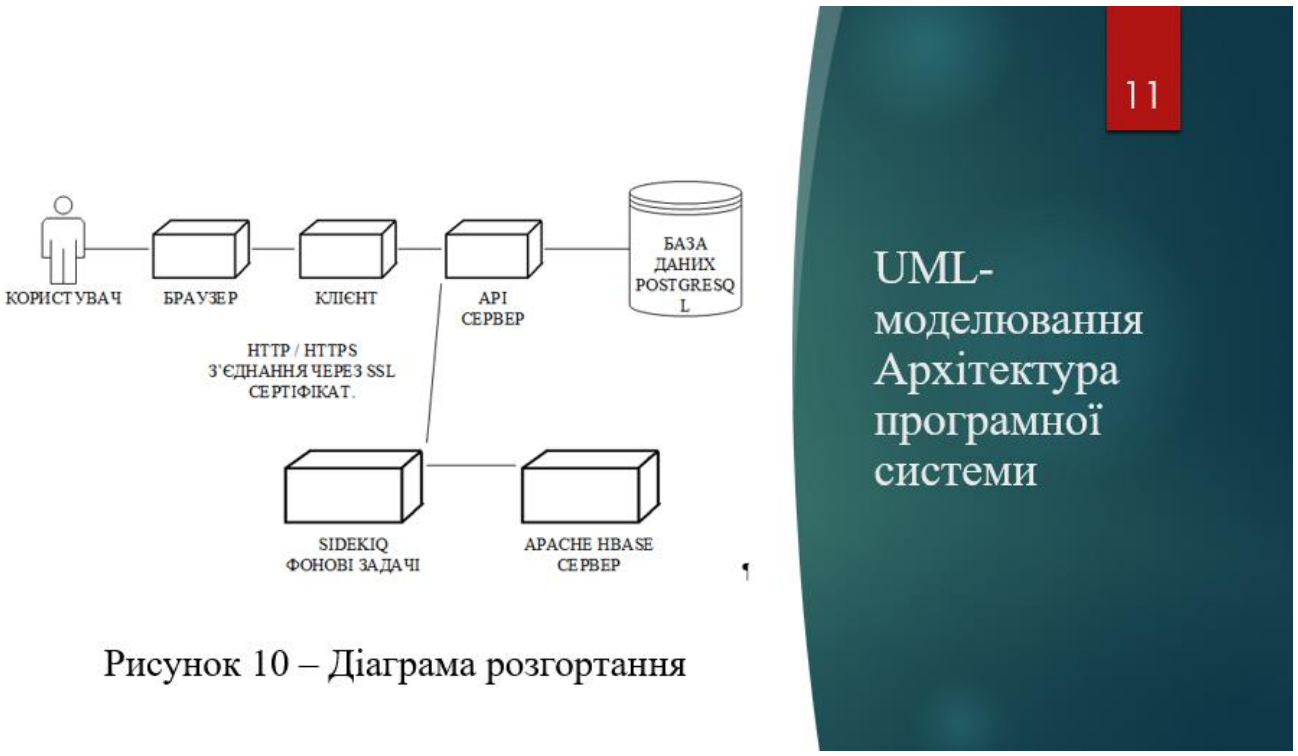


Рисунок А11 – Слайд презентації № 11

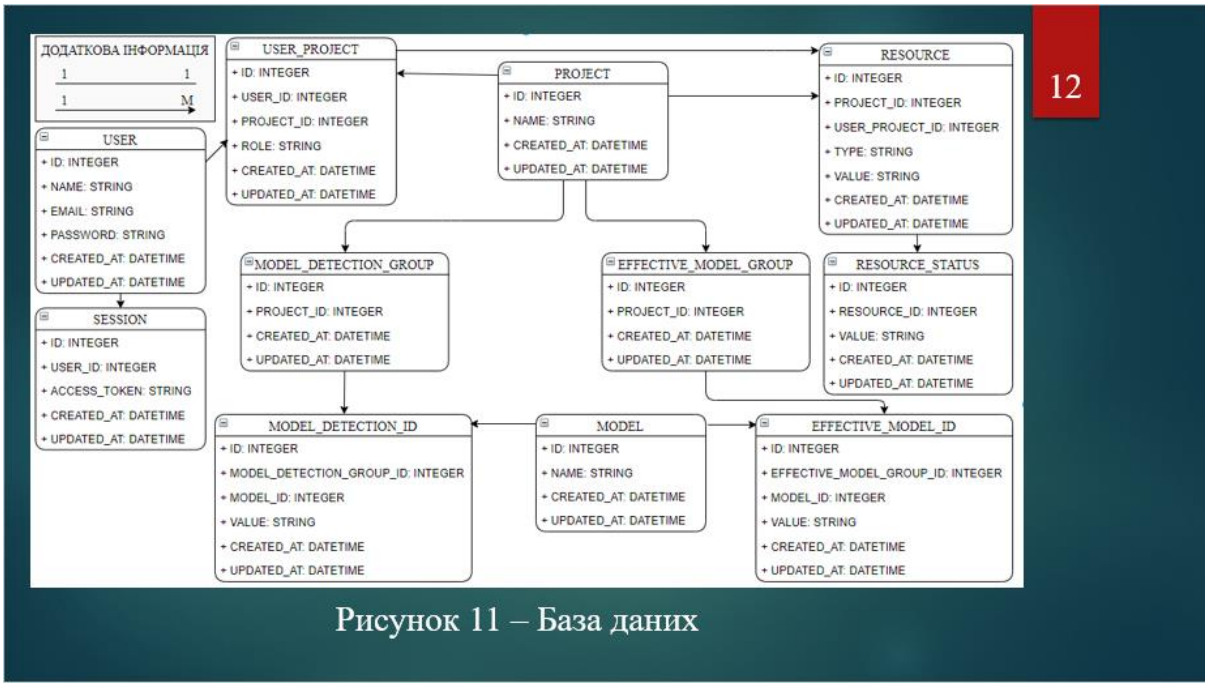


Рисунок А12 – Слайд презентації № 12

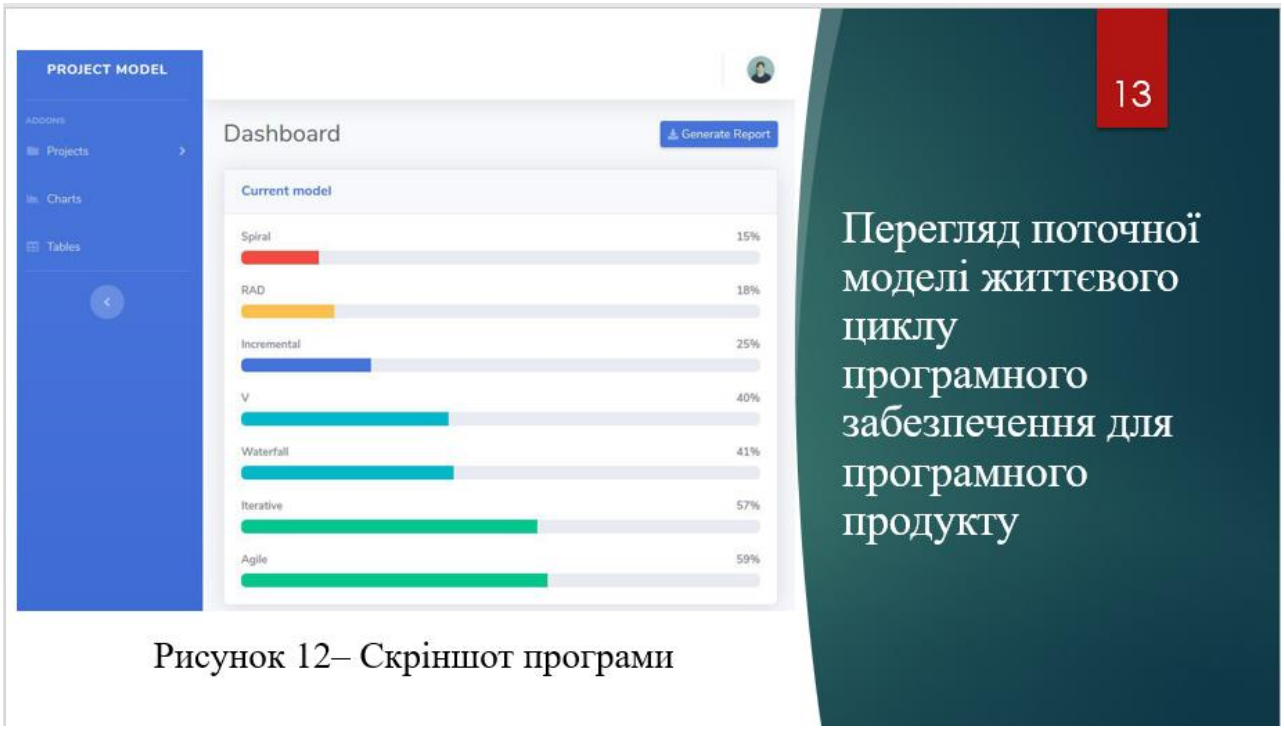


Рисунок 12– Скріншот програми

Рисунок А13 – Слайд презентації № 13

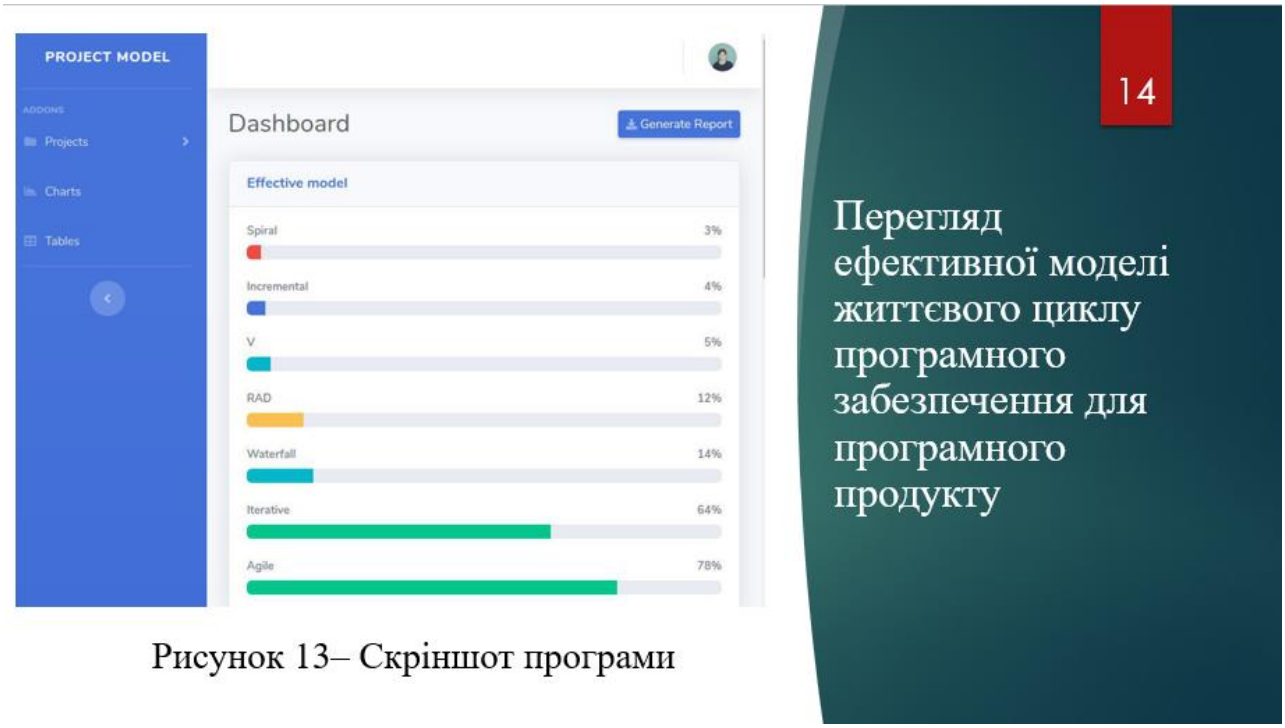


Рисунок 13– Скріншот програми

Рисунок А14 – Слайд презентації № 14


Рисунок 14 – Категорія 1

Рисунок А15 – Слайд презентації № 15


Рисунок 15 – Категорія 2

Рисунок А16 – Слайд презентації № 16

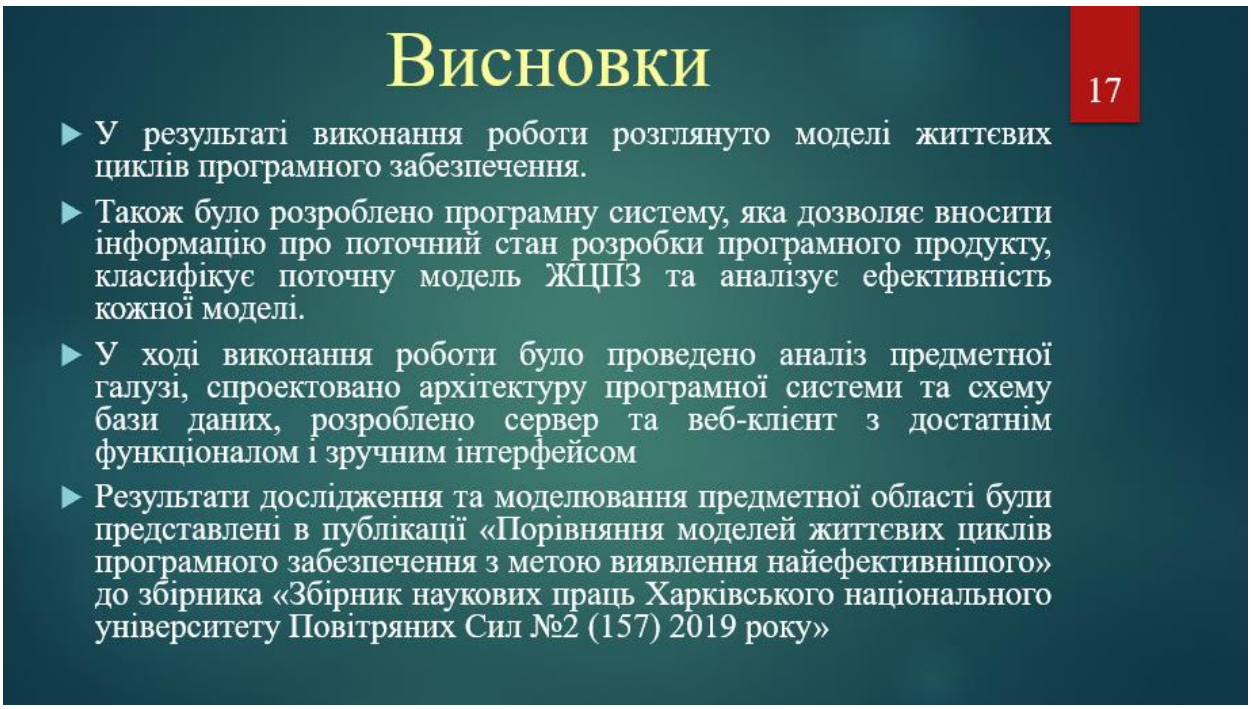


Рисунок А17 – Слайд презентації № 17

# ДОДАТОК Б

## ЛІСТИНГ КОДУ ПРОГРАМИ

```
source 'https://rubygems.org'
git_source(:github) { |repo| "https://github.com/#{repo}.git" }

ruby '2.5.1'

# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '~> 5.2.2', '>= 5.2.2.1'
# Use postgresql as the database for Active Record
gem 'pg', '>= 0.18', '< 2.0'
# Use Puma as the app server
gem 'puma', '~> 3.11'
# Use SCSS for stylesheets
gem 'sass-rails', '~> 5.0'
# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'
# See https://github.com/rails/execjs#readme for more supported runtimes
# gem 'mini_racer', platforms: :ruby

# Use CoffeeScript for .coffee assets and views
gem 'coffee-rails', '~> 4.2'
# Turbolinks makes navigating your web application faster. Read more:
https://github.com/turbolinks/turbolinks
gem 'turbolinks', '~> 5'
# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 2.5'
# Use Redis adapter to run Action Cable in production
# gem 'redis', '~> 4.0'
# Use ActiveModel has_secure_password
gem 'bcrypt', '~> 3.1.7'

# Use ActiveStorage variant
# gem 'mini_magick', '~> 4.8'

# Use Capistrano for deployment
# gem 'capistrano-rails', group: :development

# Reduces boot times through caching; required in config/boot.rb
gem 'bootsnap', '>= 1.1.0', require: false

gem 'dotenv-rails'
gem 'faker'
gem 'ruby_linear_regression'
gem 'rb-libsvm'
gem 'classifier-reborn'

group :development, :test do
  # Call 'byebug' anywhere in the code to stop execution and get a debugger
console
  gem 'byebug', platforms: [:mri, :mingw, :x64_mingw]
end

group :development do
  # Access an interactive console on exception pages or by calling 'console'
anywhere in the code.
  gem 'web-console', '>= 3.3.0'
```

```
    gem 'listen', '>= 3.0.5', '< 3.2'
    # Spring speeds up development by keeping your application running in the
background. Read more: https://github.com/rails/spring
    gem 'spring'
    gem 'spring-watcher-listen', '~> 2.0.0'
  end


  # Windows does not include zoneinfo files, so bundle the tzinfo-data gem
  gem 'tzinfo-data', platforms: [:mingw, :mswin, :x64_mingw, :jruby]
  # This file is auto-generated from the current state of the database. Instead
  # of editing this file, please use the migrations feature of Active Record to
  #  incrementally modify your database, and then regenerate this schema
definition.
  #
  # Note that this schema.rb definition is the authoritative source for your
  # database schema. If you need to create the application database on another
  # system, you should be using db:schema:load, not running all the migrations
  # from scratch. The latter is a flawed and unsustainable approach (the more
migrations
  # you'll amass, the slower it'll run and the greater likelihood for issues).
  #
  # It's strongly recommended that you check this file into your version control
system.

  ActiveRecord::Schema.define(version: 2019_05_12_150941) do

    # These are extensions that must be enabled in order to support this database
    enable_extension "plpgsql"

    create_table "detection_models", force: :cascade do |t|
      t.bigint "model_detection_group_id"
      t.bigint "model_id"
      t.string "value"
      t.datetime "created_at", null: false
      t.datetime "updated_at", null: false
      t.index                 ["model_detection_group_id"],                 name:
"index_detection_models_on_model_detection_group_id"
      t.index ["model_id"], name: "index_detection_models_on_model_id"
    end

    create_table "effective_model_groups", force: :cascade do |t|
      t.bigint "project_id"
      t.datetime "created_at", null: false
      t.datetime "updated_at", null: false
      t.index                       ["project_id"],                       name:
"index_effective_model_groups_on_project_id"
    end

    create_table "effective_models", force: :cascade do |t|
      t.bigint "effective_model_group_id"
      t.bigint "model_id"
      t.string "value"
      t.datetime "created_at", null: false
      t.datetime "updated_at", null: false
      t.index                 ["effective_model_group_id"],                 name:
"index_effective_models_on_effective_model_group_id"
      t.index ["model_id"], name: "index_effective_models_on_model_id"
    end

    create_table "model_detection_groups", force: :cascade do |t|
      t.bigint "project_id"
      t.datetime "created_at", null: false
      t.datetime "updated_at", null: false
```

```
      t.index                    ["project_id"],                    name:
"index_model_detection_groups_on_project_id"
    end

    create_table "models", force: :cascade do |t|
      t.string "name"
      t.datetime "created_at", null: false
      t.datetime "updated_at", null: false
    end

    create_table "projects", force: :cascade do |t|
      t.string "name"
      t.datetime "created_at", null: false
      t.datetime "updated_at", null: false
    end

    create_table "resource_statuses", force: :cascade do |t|
      t.bigint "resource_id"
      t.string "value"
      t.index ["resource_id"], name: "index_resource_statuses_on_resource_id"
    end

    create_table "resources", force: :cascade do |t|
      t.bigint "project_id"
      t.bigint "user_project_id"
      t.string "resource_type"
      t.string "value"
      t.datetime "created_at", null: false
      t.datetime "updated_at", null: false
      t.index ["project_id"], name: "index_resources_on_project_id"
      t.index ["user_project_id"], name: "index_resources_on_user_project_id"
    end

    create_table "sessions", force: :cascade do |t|
      t.bigint "user_id"
      t.string "access_token"
      t.datetime "created_at", null: false
      t.datetime "updated_at", null: false
      t.index ["user_id"], name: "index_sessions_on_user_id"
    end

    create_table "user_projects", force: :cascade do |t|
      t.bigint "user_id"
      t.bigint "project_id"
      t.string "role"
      t.datetime "created_at", null: false
      t.datetime "updated_at", null: false
      t.index ["project_id"], name: "index_user_projects_on_project_id"
      t.index ["user_id"], name: "index_user_projects_on_user_id"
    end

    create_table "users", force: :cascade do |t|
      t.string "email"
      t.string "name"
      t.string "password_digest"
      t.datetime "created_at", null: false
      t.datetime "updated_at", null: false
    end
  add_foreign_key "detection_models", "model_detection_groups"
    add_foreign_key "detection_models", "models"
    add_foreign_key "effective_model_groups", "projects"
    add_foreign_key "effective_models", "effective_model_groups"
    add_foreign_key "effective_models", "models"
    add_foreign_key "model_detection_groups", "projects"
```

```
  add_foreign_key "resource_statuses", "resources"
  add_foreign_key "resources", "projects"
  add_foreign_key "resources", "user_projects"
  add_foreign_key "sessions", "users"
  add_foreign_key "user_projects", "projects"
  add_foreign_key "user_projects", "users"
end
class EffectiveModel < ApplicationRecord
  belongs_to :model
  belongs_to :effective_model_group

  validates :value, presence: true
end
class EffectiveModelGroup < ApplicationRecord
  has_many :effective_models, dependent: :destroy

  belongs_to :project
end
class Model < ApplicationRecord
  has_many :effective_models, dependent: :destroy
  has_many :model_detections, dependent: :destroy

  validates :name, presence: true, uniqueness: true
end
class ModelDetection < ApplicationRecord
  belongs_to :model
  belongs_to :model_detection_group

  validates :value, presence: true
end
class ModelDetectionGroup < ApplicationRecord
  has_many :model_detections, dependent: :destroy

  belongs_to :project
end
class Project < ApplicationRecord
  has_many :user_projects
  has_many :users, through: :user_projects

  validates :name, presence: true
end
class Resource < ApplicationRecord
  enum resource_type: {
    requirements: 'requirements',
    planning: 'planning',
    capital: 'capital',
    developers: 'developers',
    tasks: 'tasks',
    bugs: 'bugs'
  }

  has_many :resource_statuses, dependent: :destroy

  belongs_to :project
  belongs_to :user_project

  validates :resource_type, presence: true
  validates :value, presence: true
end
class ResourceStatus < ApplicationRecord
  belongs_to :resource

  validates :value, presence: true
end
```

```
class Session < ApplicationRecord
  belongs_to :user

  validates :access_token, presence: true, uniqueness: { scope: :user }

  before_save :generate_access_token

  private

  def generate_access_token
    self.access_token = loop do
      random_token = SecureRandom.urlsafe_base64(nil, false)
      break random_token unless Session.exists?(access_token: random_token)
    end
  end
end
class User < ApplicationRecord
  has_secure_password

  has_many :sessions, dependent: :destroy
  has_many :user_projects, dependent: :destroy
  has_many :projects, through: :user_projects

  validates :email, presence: true, uniqueness: true
  validates :name, presence: true
end
class UserProject < ApplicationRecord
  enum role: {
    admin: 'admin',
    scrum_master: 'scrum_master'
  }

  belongs_to :user
  belongs_to :project

  validates :role, presence: true
end
```

# ДОДАТОК В

## АПРОБАЦІЯ РЕЗУЛЬТАТІВ РОБОТИ

УДК 004.051 DOI: 10.30748/

В.В. Голян, О.К. Кравченко[2]

[1]*Харківський національний університет радіоелектроніки, Харків*
[2]*Харківський національний університет радіоелектроніки, Харків*

### ПОРІВНЯННЯ МОДЕЛЕЙ ЖИТТЄВИХ ЦИКЛІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З МЕТОЮ ВИЯВЛЕННЯ НАЙЕФЕКТИВНІШОГО

*У статті обґрунтована актуальність дослідження моделей життєвих циклів програмного забезпечення. Проведен аналіз таких моделей життєвих циклів програмного забезпечення, як модель водоспаду, V-модель, інкрементаційна модель, RAD модель, ітераційна модель, Agile модель, спіральна модель. А також спроектована архітектура програмної системи та схеми бази даних для програмного продукту, який має збирати інформацію про моделі життєвих циклів, за допомогою якої можна порівняти їх. У роботі розглянуто порівняння сьоми моделей за допомогою розробки двох різних категорій програмних продуктів двома командами розробки з однаковим штатним складом.*

***Ключові слова:*** *модель, життєвий цикл, програмне забезпечення, програмний продукт.*

### Вступ

**Постановка проблеми.** Процес розробки програмного забезпечення є структурою, що накладається на розробку програмного продукту.

Процес є основоположним інструментом для досягнення консенсусу суспільства та сприяння дуже великої кількості людей для роботи над спільним проектом. Методичний підхід до розробки програмного забезпечення призводить до меншої кількості дефектів і, отже, у кінцевому рахунку забезпечує більш короткий термін випуску та кращу вартість. Необхідність вибору та слідування процесу розробки програмного забезпечення полягає в забезпеченні бажаної дисципліни для створення якісного продукту для успішного ведення бізнесу та для уникнення втрати часу, грошей, деморалізації в розробниках тощо.

Життєвий цикл розробки програмного забезпечення стикається з багатьма проблемами на кожному етапі. Найгірші ситуації – це запуск проекту з новими співробітниками, які не мають досвіду в галузі, не мають доказових технологій, а також мають складний термін. Поряд з технічними викликами будь-якої ситуації можуть перешкодити розробці програмного забезпечення і поставити управління в ризиковану і страшну кризу, яка не вирішила добре цю ситуацію, може призвести до того, що – продукти перевищують як вартість, так і оцінку часу, але все ще закінчуються поганою якістю. Вони не відповідають технічним вимогам, визначеним споживачем, і, нарешті, призводять до відмови бізнесу.

Ціллю даної роботи є проведення глибокого аналізу та порівняння різноманітних моделей життєвого циклу програмного забезпечення, дослідження процесу вибору найефективнішою моделі в залежності від базових характеристик і ресурсів, яка має команда до початку створення продукту.

Дана робота допомагає проаналізувати питання, які стосуються вибору моделі життєвого циклу програмного забезпечення на початку, та допомагає актуально підібрати можливі альтернативи в підходах та організації циклу програмного забезпечення в процесі розробки.

**Аналіз останніх досліджень і публікацій.** У роботі [3] досліджено ефективність моделей життєвих циклів програмного забезпечення для процесів комунікації та взаємопраці між командою розробників та замовниками. У роботі [4] досліджено існуючи моделі життєвих циклів програмного забезпечення та виявлені категоріх програмних продуктів для порівняння.

**Мета статті** – проведення глибокого аналізу та порівняння сьоми основних моделей життєвих циклів програмного забезпечення, вибір найефективнішої моделі.

### Виклад основного матеріалу

Життєвий цикл розробки програмного забезпечення, коротко кажучи, ЖЦПЗ – це чітко визначена, структурована послідовність етапів розробки програмного забезпечення для розробки призначеного програмного продукту.

ЖЦПЗ визначає повний цикл розвитку, тобто всі завдання, пов'язані зі збором вимоги щодо обслуговування продукту.

ЖЦПЗ надає ряд кроків, які необхідно дотримуватися для ефективного проектування та розробки програмного продукту. Структура ЖЦПЗ включає в себе етапи програмного забезпечення: спілкування, збір вимог, техніко-економічне обґрунтування, системний аналіз, дизайн програмного забезпечення, кодування, тестування,

інтеграція, реалізація, експлуатація та обслуговування, диспозиція.

Модель водоспаду розвиває програмне забезпечення поступово (рис. 1): операційний аналіз, експлуатаційні специфікації, специфікації проектування і кодування, розробка, тестування, розгортання, оцінка.

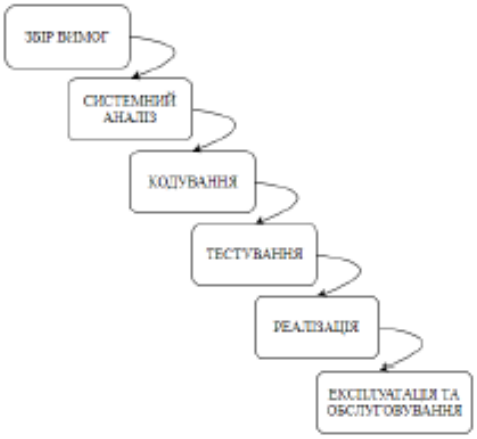


Рис. 1. Модель водоспаду

V-модель є варіацією моделі водоспаду у V-формі, складеної навпіл на найнижчому рівні розкладання (рис. 2). Ліва нога V-форми представляє еволюцію вимог користувача до все більш дрібних компонентів через процес декомпозиції і визначення; Права нога являє інтеграцію та перевірку системних компонентів у послідовні рівні впровадження та складання. Вертикальна вісь зображує рівень розкладання від системного рівня, у верхній частині, до найнижчого рівня деталізації на рівні компонентів внизу.

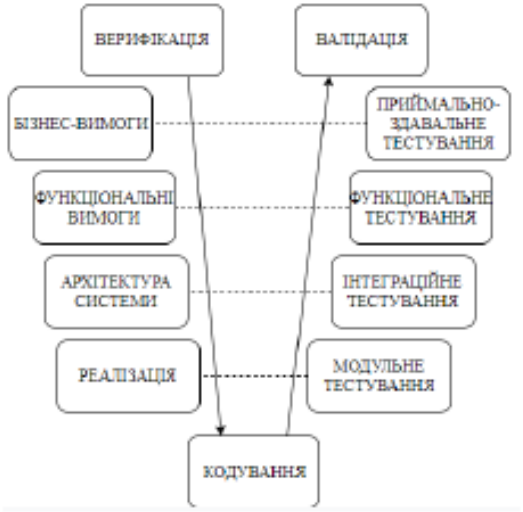


Рис. 2. V-модель

Інкрементальну модель можна розглядати як тривимірне представлення моделі водоспаду (рис. 3).

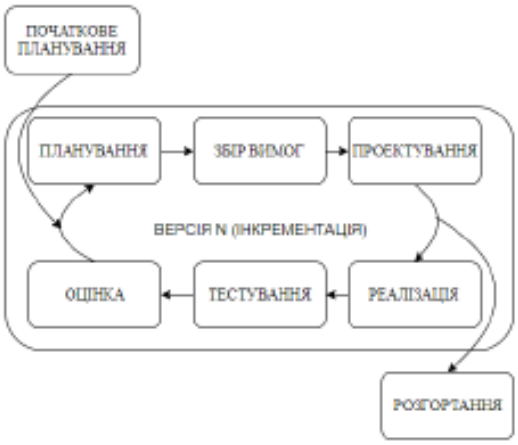


Рис. 3. Інкрементаційна модель

Модель RAD – модель швидкого розробки додатків. Це тип інкрементної моделі. В моделі RAD компоненти або функції розробляються паралельно, як якщо б вони були міні-проектами. Розробки є коробкою часу, доставляються і потім збираються в робочий прототип (рис. 4).

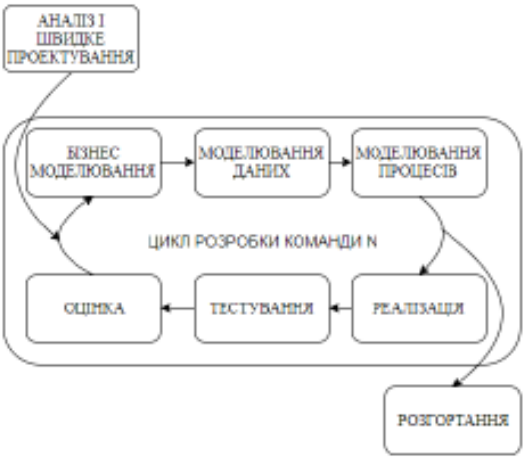


Рис. 4. Модель RAD

Модель розробки програмного забезпечення Agile була призначена головним чином для того, щоб допомогти розробникам створити проект, який може швидко адаптуватися до перетворення запитів. Отже, найважливішим завданням для розробки моделі Agile є спрощення та швидке досягнення проекту. Для досягнення цього завдання розробникам необхідно зберегти спритність під час розробки. Спритність може бути досягнута шляхом коригування прогресу в проекті шляхом усунення

діяльності, яка може не мати вирішального значення для цього конкретного проекту

Для кожної ітерації залучається міжфункціональна група розробників, що працюють одночасно в різних областях розробки продукту (рис. 5), таких як: планування, аналіз вимог, дизайн, розвиток, тестування одиниць, розгортання.

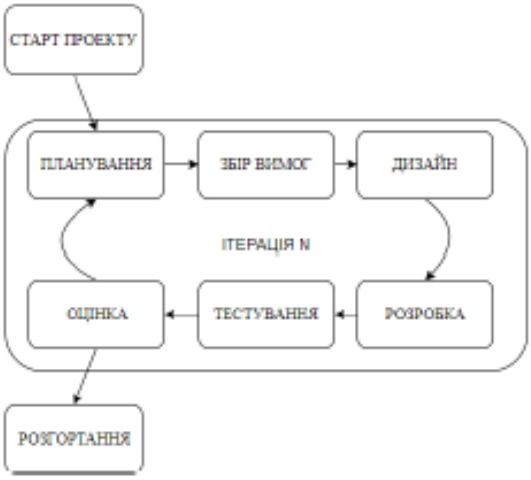


Рис. 5. Модель Agile

У ітеративній моделі ітераційний процес починається з простої реалізації невеликого набору програмних вимог і ітераційно посилює нові версії, поки не буде реалізована і готова до розгортання повна система (рис. 6).


Рис. 6. Ітераційна модель

Модель спіралі являє собою комбінацію як ітеративної моделі, так і однієї моделі ЖЦПЗ. Це можна побачити, як якщо б ви обрали одну модель ЖЦПЗ і об'єднали її з циклічним процесом (рис. 7).

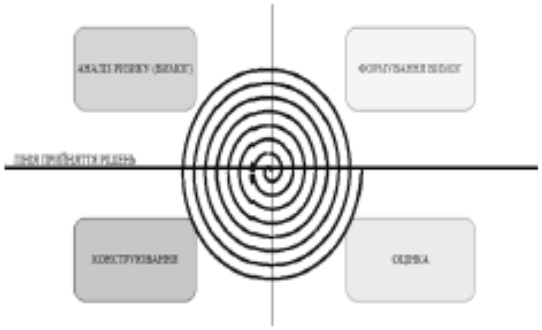


Рис. 7. Спіральна модель

Модель спіралі являє собою комбінацію як ітеративної моделі, так і однієї моделі ЖЦПЗ. Це можна побачити, як якщо б ви обрали одну модель ЖЦПЗ і об'єднали її з циклічним процесом (рис. 7).

Для порівняння обраних моделей життєвого циклу програмного забезпечення слід обрати такі критерії порівняння, які будуть найкраще демонструвати переваги та недоліки кожної. Критерії порівняння мають продемонструвати, які моделі життєвого циклу програмного забезпечення краще використовувати в яких ситуаціях і які в них є можливості.

Приведемо перелік обраних критеріїв порівняння моделей життєвого циклу програмного забезпечення:

– час реалізації;

– вартість реалізації;

– категорія програмного забезпечення.

Час реалізації програмного продукту одначасно може впливати на його вартість та на актуальність на ринку програмних продуктів. Швидкий вихід на ринок дозволить програмному продукту заробити набагато більше в перший час. Можна сказати, що час реалізації обернено пропорційний добутку програмного продукту.

Вартість реалізації впливає на якість програмного продукту, на якість обслуговування та на якість майбутньої підтримки. Склад команди розробників залежить від можливостей капіталу програмного продукту. Можна сказати, що вартість реалізації обернено пропорційне часу реалізації програмного продукту.

Розглянемо основні етапи моделей життєвих циклів програмного забезпечення:

– збір вимог – етап, на якому команда розробників програмного забезпечення працює над виконанням проекту, проводить обговорення з різними зацікавленими сторонами з проблемних областей і намагається виявити якомога більше інформації щодо їх вимог;

– конструювання (кодування) – етап, на якому проводиться реалізація розробки програмного забезпечення починається з точки зору написання програмного коду на відповідній мові програмування та ефективного розробки виконуваних програм без помилок;

– тестування – етап, на якому може тестуватися результат конструювання та результат збору вимог або системного аналізу програмного продукту;

– оцінка – етап, на якому проводиться аналіз отриманих результатів з усіх попередніх етапів та будується наступний план розвитку системи;

– розгортання – етап, на якому проводиться інтеграція програмного забезпечення з об'єктами зовнішнього світу та реалізація продукту для доступу користувачів.

Основні етапи моделей життєвих циклів програмного забезпечення впливають на показники часу та вартості виконання.

Таблиця 1

Вплив основних етапів на показники

| Етап життєвого циклу ПЗ | Вартість реалізації | Час реалізації |
|---|---|---|
| Збір вимог | Пропорційне | Обернено пропорційне |
| Конструювання | Пропорційне | Пропорційне |
| Тестування | Пропорційне | Пропорційне |
| Оцінка | Пропорційне | Обернено пропорційне |
| Розгортання | Пропорційне | Обернено пропорційне |

Допускається розташування великих рисунків, формул та таблиць в одну колонку (до 16,5 см).

Маємо, що велика вартість збору вимог, оцінки результатів та розгортання продукту окупаються швидкої реалізацією та швидким виходом на ринок.

Можемо виділити дві основні категорії програмного забезпечення:

– категорія 1 – упор на швидкість та якість роботи програми та документацію;

– категорія 2 – упор на зовнішній вигляд інтерфейсу, зручність використання.

Тобто маємо 2 абсолютно різні категорії, де перша має дуже якісні внутрішні характеристики, а друга має зовнішні характеристики.

На старті маємо 2 ідентичні команди розробки, які складаються з двох програмистів, одного тестувальника, одного досвідченого архітектора та одного бізнес-аналітика.

Командам потрібно релізувати програмний продукт, для якого потрібно розробити серверну частину для автентифікації та авторизації користувачів, на кожній з 7-ми основних моделей життєвих циклів програмного забезпечення: моделі водопаду, V-моделі, RAD-моделі, ітераційної моделі, ітеративної моделі, Agile моделі та спіральної моделі.

Задачі розробки програмного продукту можна розбити на 4 частини:

– налаштування проекту;

– створення функціоналу користувачів;

– створення функціоналу для сесій користувачів;

– налаштування API ендпойнтів для реєстрації та автентифікації.

Для ще більшого наближення до реального світу маємо дві умови для кожної моделі:

– тестування API ендпойнтів приводить до знаходження дефектів у системі;

– після вдалого тестування всіх базових умов треба змінити вимоги до програмного продукту.

Кожний етап будемо обчислювати за допомогою витраченого часу в хвилинах та коштів, виділених на цей етап.

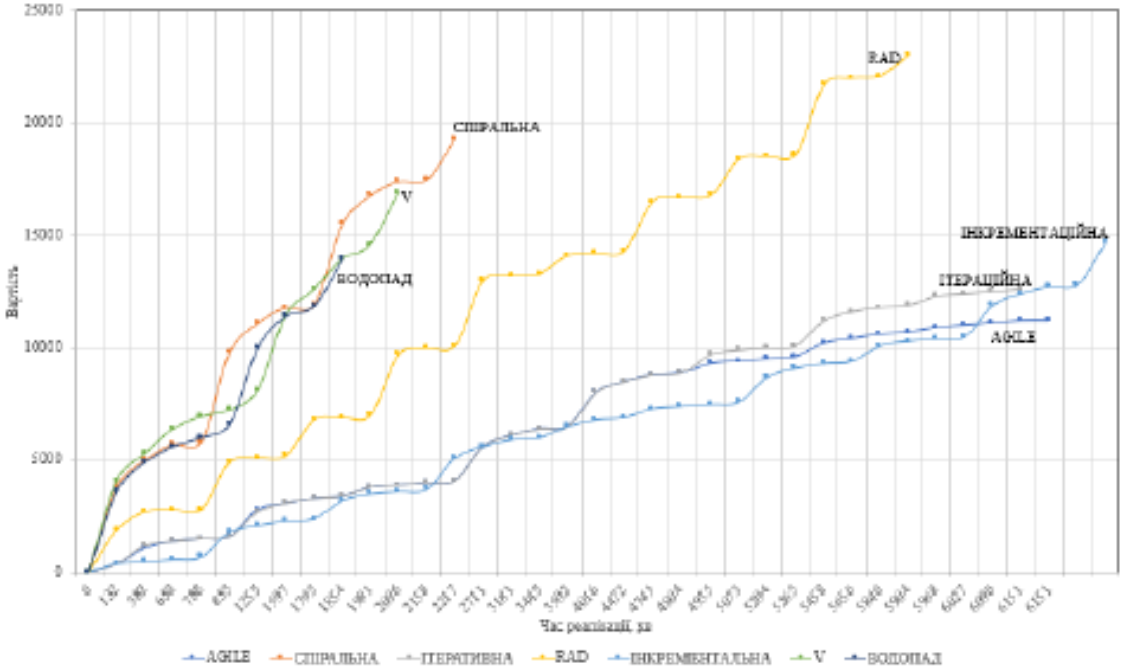


Рис. 8 Графік розробки для першої категорії

Команди практично однаково закінчили розробку програмного продукту, тому для кінцевого аналізу достатньо зрівнювати за даними будь-якої команди. Побудуємо графік всіх моделей життєвого циклу програмного забезпечення для першої команди для результатів розробки програмного продукту першої категорії (рис.8).

Таблиця 2 зображає загальний аналіз, який показує, що процес розробки для кожної моделі має практично ідентичну фігуру, але можна виявити кращі моделі по показникам загальної вартості та часу розробки.

Таблиця 2

Аналіз показників

| Модель | Загальний час, хв | Загальна вартість | Вартість / Час, 1 / хв |
|---|---|---|---|
| Водопад | 7717 | 13950 | 1,81 |
| V | 10231 | 16900 | 1,65 |
| Інкрементна | 8201 | 14700 | 1,79 |
| RAD | 9163 | 23000 | 2,51 |
| Ітеративна | 7013 | 12600 | 1,80 |
| Спіральна | 10469 | 19300 | 1,84 |
| Agile | 6151 | 11200 | 1,82 |

Для другої категорії на старті маємо 2 ідентичні команди розробки, які складаються з двох програмистів, одного тестувальника, одного дизайнера, одного досвідченого архітектора та одного бізнес-аналітика.

Командам потрібно релізувати програмний продукт, для якого потрібно розробити клієнтську частину для зображення дій користувача (реєстрації та автентифікації), на кожній з 7-ми основних моделей життєвих циклів програмного забезпечення: моделі водопаду, V-моделі, RAD-моделі, ітераційної моделі, ітеративної моделі, Agile моделі та спіральної моделі.

Задачі розробки програмного продукту можна розбити на 4 частини:

– дизайн сторінок;

– налаштування проекту;

– додавання статичного функціоналу після дизайну (верстка);

– додавання динамічного функціоналу (використання js фреймворку).

Для ще більшого наближення до реального світу маємо дві умови для кожної моделі:

– зміна дизайну сторінок від замовника 2 рази;

– зміна дизайну після реалізації функціоналу.

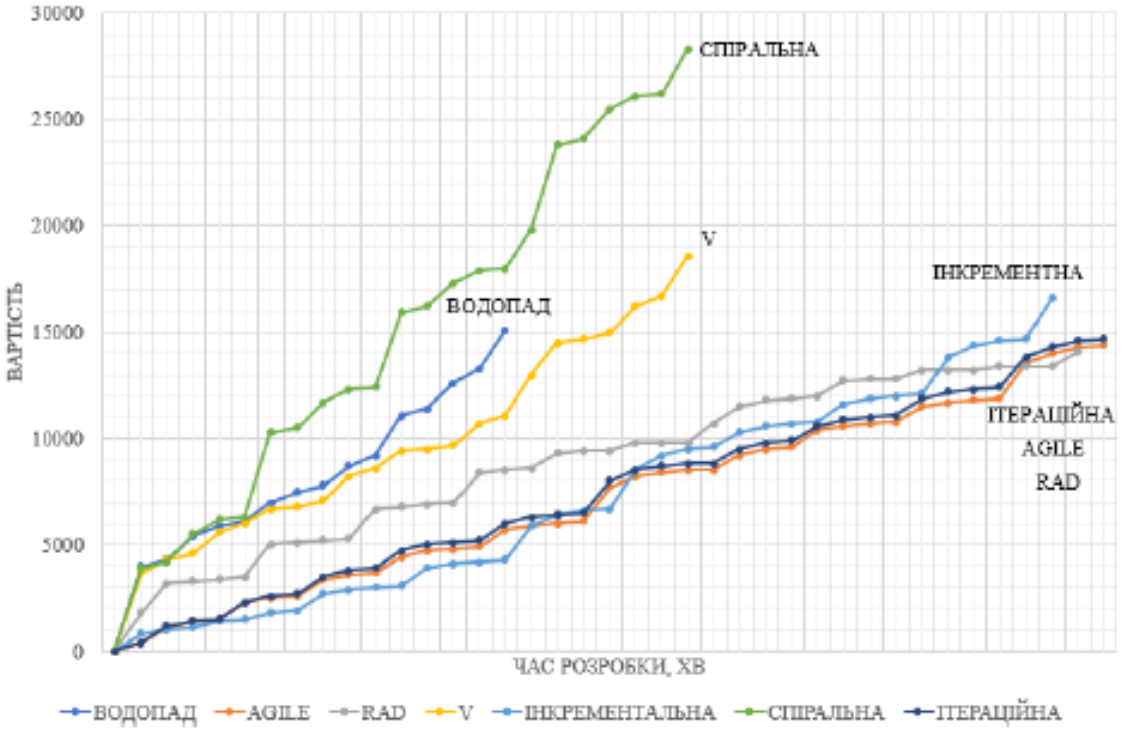


Рис. 9 Графік розробки для другої категорії

Можна зробити висновок, що команди практично однаково закінчили розробку програмного продукту, тому для кінцевого аналізу достатньо зрівнювати за даними будь-якої команди. Побудуємо графік всіх моделей життєвого циклу програмного забезпечення для першої команди для результатів розробки програмного продукту першої категорії. На рисунку 9 зображено загальний аналіз усіх моделей для ПЗ категорії 2.

Для розробки програмного продукту з упором на швидкість та якість роботи програми та документацію порівняємо всі приведені моделі по двом характеристикам: часу та вартості розробки. Тенденція розробки практично однакова для всіх моделей.

Якщо час розробки є найважливішим критерієм успіху програмного продукту, то слід вибирати модель життєвого циклу програмного забезпечення в такому порядку, як Agile, ітеративна, водопад, інкрементна, RAD, V, спіральна моделі. Зазначимо, що Agile очіковано краще за ітеративну модель, а V-модель неочіковано гірше за водопад.

## Висновки

Для розробки програмного продукту з упором на швидкість та якість роботи програми та документацію порівняємо всі приведені моделі по двом характеристикам: часу та вартості розробки. Тенденція розробки практично однакова для всіх моделей.

Якщо час розробки є найважливішим критерієм успіху програмного продукту, то слід вибирати модель життєвого циклу програмного забезпечення в такому порядку, як Agile, ітеративна, водопад, інкрементна, RAD, V, спіральна моделі. Зазначимо, що Agile очіковано краще за ітеративну модель, а V-модель неочіковано гірше за водопад.

Якщо вартість розробки є найважливішим критерієм успіху програмного продукту, то вибирати модель життєвого циклу програмного забезпечення в такому порядку, як Agile, ітеративна, водопад, інкрементна, RAD, V, спіральна моделі. Тільки RAD модель має непропорційно велеки вартість розробки.

Для розробки програмного продукту з упором на зовнішній вигляд інтерфейсу та зручне використання порівняємо всі приведені моделі по двом характеристикам: часу та вартості розробки. Тенденція розробки практично однакова для всіх моделей, рис. 9.

Якщо час розробки є найважливішим критерієм успіху програмного продукту, то слід вибирати модель життєвого циклу програмного забезпечення в такому порядку, як RAD, ітеративна, Agile, ітеративна, інкрементна, водопад, V, спіральна моделі. Зазначимо, що RAD для другої категорії є найкращою моделею, а для першої результат зворотній.

Якщо вартість розробки є найважливішим критерієм успіху програмного продукту, то вибирати модель життєвого циклу програмного забезпечення в такому порядку, як RAD, Agile, ітеративна, водопад, інкрементна, V, спіральна. Результат для двох показників ідентичний.

В загальному випадку для категорії 1 портрібно розглядати тільки такі моделі, як Agile, ітеративну модель та водопад. Для категорії 2 потрібно розглядати тільки RAD, Agile, ітеративну та водопад.

На початку проекту, коли вибирається модель життєвого циклу програмного забезпечення, не завжди можливо визначити, який тип програмного продукту потрібно реалізувати. В такому випадку треба дивитись на стабільність моделі життєвого циклу програмного забезпечення та залежність від категорій програмних продуктів.

Проведені дослідження показали, що порядок вибору моделі життєвого циклу програмного забезпечення буде наступним: Agile модель, ітеративна модель, модель водопаду, RAD, V-модель, інкрементна модель та спіральна модель. Варто зазначити, що даний порядок зменшує ризики втратити час та збільшити витрати на розробку.

Було виявлено, що для категорії програмних продуктів з упором на швидкість на документацію потрібно використовувати такі моделі, як Agile, ітеративну та водопад. Для категорії програмних продуктів з упором на користувальницький інтерфейс та зручність використання потрібно дивитись на такі моделі, як RAD, Agile, ітеративну та водопад. У загальному випадку Agile модель має найменші ризики втратити більше часу на розробку, найбільші ризики має спіральна модель.

Отже, були виконані усі вимоги, поставлені до роботи. Є вся необхідна теоретична база для розробки та порівняння. Проведено аналіз та порівняння лідерів, які виявляються практично однаковими за різних критеріїв.

## Список літератури

1. Модели жизненного цикла / П. Добряк. – СПб.: Символ-Плюс, 2016. – 132 с.
2. Системная инженерия. Принципы и практика / А. Косяков. – СПб.: ДМК-Пресс, 2017. – 624 с.
3. A Software System Development Life Cycle Model for Improved Stakeholders' Communication and Collaboration / S. Cohen, D. Dori, U. de Haan // Int. J. of Computers, Communications & Control. – 2010. – Vol. V, No. 1. – P. 20. https://doi.org/10.15837/ijccc.2010.1.2462.
4. Software Development Lifecycle Models / B.R. Nayan // ACM SIGSOFT Software Engineering Notes. – 2010. – Vol. 35, No. 3. – P. 8. https://doi.org/10.1145/1764810.1764814.
5. A Survey on Software Development Life Cycle Models / T. Bhuvaneswar, S. Prabaharan // International Journal of Computer Science and Mobile Computing. – 2013. – Vol. 2, Issue. 5. – Pp. 262-267.
6. Software Development Life Cycle Models Comparison, Consequences / V. Rastogi. // International Journal of Computer Science and Information Technologies. – 2015. – Vol. 6, No. 1. – P. 168.
7. A Simulation Model for the Waterfall Software Development Life Cycle / Y. Bassil. // International Journal of Engineering & Technology (iJET). – 2012. –Vol. 2, No. 5. – P. 16.
8. A detailed study of Software Development Life Cycle (SDLC) Models / S. Barjtya, A. Sharma, U. Rani // International Journal Of Engineering And Computer Science. – 2017. – Vol. 6, Issue. 7. – P. 22097.
9. A Genetic Based Intelligent Approach to Estimate Software Release Using Agile / P. Rana, N. Bilandi // International Journal of Advanced Research in Computer Science and Software Engineering. – 2012. – Vol. 2, Issue. 8. – P. 311.
10. Software development as a service: agile experiences. / Lehman, J. Tobin, A. Sharma // SRII Global Conference (SRII), Annual, IEEE. – 2011. https://doi.org/10.1109/srii.2011.82.
11. Agile software development: Impact on productivity and quality / A. Ahmed // Management of Innovation and Technology (ICMIT), IEEE International Conference. – 2010. https://doi.org/10.1109/icmit.2010.5492703.
12. About Software Engineering Frameworks and Methodologies / M. Ernest // IEEE AFRICON. – 2009. https://doi.org/10.1109/afrcon.2009.5308117.

## References

1. Dobryak, P. (2016) "Modely zhyznennogo cykla" [The lifecycle models], Symvol-Plus, Saint Petersburg, 132 p.
2. Kosyakov, A. (2017) "Systemnaya inzheneryiya. Pryncypi y praktyka" [System engineering. Principles and practice], DMK-Press, Saint Petersburg, 624 p.
3. Cohen, S., Dori, D. and de Haan Cohen, U. (2010) A Software System Development Life Cycle Model for Improved Stakeholders Communication and Collaboration, Int. J. of Computers, Communications & Control, Vol. V, No. 1, p. 20. https://doi.org/10.15837/ijccc.2010.1.2462.
4. Ruparelia, N.B. (2010) Software Development Lifecycle Models, ACM SIGSOFT Software Engineering Notes, Vol. 35, No. 3, p. 8. https://doi.org/10.1145/1764810.1764814.
5. Bhuvaneswar, T. and Prabaharan, S. (2013) A Survey on Software Development Life Cycle Models, International Journal of Computer Science and Mobile Computing, Vol. 2, Issue. 5, pp. 262-267.
6. Rastogi, V. (2015) Software Development Life Cycle Models Comparison, Consequences, International Journal of Computer Science and Information Technologies, Vol. 6, No. 1, p. 168.
7. Bassil, Y. (2012) A Simulation Model for the Waterfall Software Development Life Cycle, International Journal of Engineering & Technology (iJET), Vol. 2, No. 5, p. 16.
8. Barjtya, S., Sharma, A. and Rani S. (2017) A detailed study of Software Development Life Cycle (SDLC) Models, International Journal Of Engineering And Computer Science, Vol. 6, Issue. 7, p. 22097.
9. Rana, P. and Bilandi, N. (2012) A Genetic Based Intelligent Approach to Estimate Software Release Using Agile, International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, Issue. 8, p. 311.
10. Lehman, Tobin, J. and Sharma A. (2011) Software development as a service: agile experiences, SRII Global Conference (SRII), Annual. IEEE. https://doi.org/10.1109/srii.2011.82.
11. Ahmed, A., Ahmad, S., Ehsan, N., Mirza, E. and Sarwar, S. Z. (2010) Agile software development: Impact on productivity and quality, Management of Innovation and Technology (ICMIT), 2010 IEEE International Conference on. IEEE. https://doi.org/10.1109/icmit.2010.5492703.
12. Mnkandla, E. (2009) About Software Engineering Frameworks and Methodologies, IEEE AFRICON. https://doi.org/10.1109/afrcon.2009.5308117.

*Надійшла до редколегії 00.00.2019*
*Схвалена до друку 00.00.2019*

***Відомості про авторів:***

**Голян Віра Володимирівна**
кандидат технічних наук, доцент кафедри Програмної інженерії Харківського національного університету радіоелектроніки,
Харків, Україна

***Information about the authors:***

**Vira Golian**
Candidate of Technical Sciences, Associate Professor
Senior Lecturer of Kharkiv National University of radioelectronics
Kharkiv, Ukraine

https://orcid.org/0000-0001-5981-4760

**Кравченко Олександр Костянтинович**
бакалавр Харківського національного університету радіоелектроніки,
Харків, Україна
https://orcid.org/0000-0001-5318-1165

https://orcid.org/0000-0001-5981-4760

**Oleksandr Kravchenko**
Bachelor of Kharkiv National University of radioelectronics,
Kharkiv, Ukraine
https://orcid.org/0000-0001-5318-1165

## СРАВНЕНИЕ МОДЕЛЕЙ ЖИЗНЕННОГО ЦИКЛА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ЦЕЛЬЮ ВЫЯВЛЕНИЯ САМОГО ЭФФЕКТИВНОГО

В.В. Голян, А.К. Кравченко

*В статье обоснована актуальность исследования моделей жизненных циклов программного обеспечения. Проведен анализ таких моделей жизненных циклов программного обеспечения, как модель водопада, V-модель, инкрементайна модель, RAD модель, итерационная модель, Agile модель, спиральная модель. А также спроектирована архитектура программной системы и схемы базы данных для программного продукта, который должен собирать информацию о модели жизненных циклов, с помощью которой можно сравнить их. В работе рассмотрены сравнения семи моделей с помощью разработки двух различных категорий программных продуктов двумя командами разработки по одинаковым штатным составом.*

***Ключевые слова:*** *модель, жизненный цикл, программное обеспечение, программный продукт.*

## COMPARISON OF MODELS OF LIFE CYCLE OF SOFTWARE WITH THE PURPOSE OF IDENTIFYING THE MOST EFFECTIVE

V. Golian, O. Kravchenko

*The article substantiates the relevance of the study of models of software life cycles. The analysis of such software life cycles as a waterfall model, V-model, incremental model, RAD model, iterative model, Agile model, and spiral model was carried out. And also the architecture of the software system and the database scheme for the software product, which is to collect information about the life-cycle model with which you can compare them, is designed. The paper considers comparisons of seven models through the development of two different categories of software products by two development teams with the same staffing structure.*

*To compare selected life cycle software models, you should select the following comparison criteria that will best show the advantages and disadvantages of each. The benchmarking benchmarks must demonstrate which software lifecycle models are best used in some situations and in which opportunities they have.*

*Here is a list of selected criteria for comparison of life cycle software models:*

*- time of implementation;*

*- the cost of realization;*

*- category of software.*

*The time of implementation of a software product can simultaneously affect its cost and relevance to the market of software products. A quick entry into the market will allow the software product to earn much more in the first time. We can say that the implementation time is inversely proportional to the product.*

*The cost of implementation affects the quality of the software product, the quality of service and the quality of future support. The composition of the developer team depends on the capabilities of the software product capital. We can say that the cost of implementation is inversely proportional to the time of implementation of the software product.*

*To develop a software product, with emphasis on the speed and quality of the program and documentation, compare all the given models to two characteristics: time and cost of development. The development trend is practically the same for all models.*

*If development time is the most important criterion for the success of a software product, then you should choose a software life cycle model in the order of Agile, iterative, waterfall, incremental, RAD, V, spiral models. Note that Agile is expected better than the iterative model, and the V-model is unknowingly worse than the waterfall.*

*If the cost of development is the most important criterion for the success of a software product, then choose a software life cycle model in the order of Agile, iterative, waterfall, incremental, V, spiral model, RAD. Only the RAD model has a disproportionately high cost of development.*

*To develop a software product with an emphasis on the look and feel of the interface and convenient use, compare all of the models given by two characteristics: time and cost of development. The development trend is practically the same for all models.*

*If development time is the critical success factor for a software product, then you should choose a software life cycle model in the same order as RAD, iterative, Agile, iterative, incremental, waterfall, V, spiral models. Note that RAD for the second category is the best model, and for the first result is reverse.*

*If the cost of development is the most important criterion for the success of a software product, then choose a software life cycle model in the order of RAD, Agile, iterative, waterfall, incremental, V, spiral. The result for the two indicators is identical.*

*In the general case, for category 1 only models such as Agile, iterative model and waterfall should be considered. For Category 2, only RAD, Agile, iterative, and waterfall should be considered.*

*At the beginning of the project, when choosing a software lifecycle model, it is not always possible to determine which type of software to implement. In this case, one must look at the stability of the software life cycle model and the dependence on the software product categories.*

*Studies have shown that the order of choosing a software life cycle model will be as follows: Agile model, iterative model, waterfall model, RAD, V-model, incremental model and spiral model. It is worth noting that this procedure reduces the risk of losing time and increasing development costs.*

***Keywords:*** *model, life cycle, software, software product.*