КОМПЬЮТЕРНЫЕ НАУКИ

УДК 621.389

ВЛИЯНИЕ ИЗМЕНЕНИЯ ЗАЦЕПЛЕНИЯ И СВЯЗНОСТИ НА СЛОЖНОСТЬ КОДА И ЕГО БЫСТРОДЕЙСТВИЕ В РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

КРАВЧЕНКО А.К., АФАНАСЬЕВА И.В.

Оценивается влияние мер сил взаимосвязанности внутри и между модулями на читабельность кода и скорость работы программного обеспечения. Для этого рассматриваются понятия связности и зацепления на уровне модульной архитектуры, сравнивается сложность кода с помощью инструмента flog, а также быстродействие обработки HTTP-запросов с помощью программного обеспечения Apache JMeteor.

Ключевые слова: программное обеспечение, связность, зацепление, читабельность кода, модульная архитектура.

Key words: software, cohesion, coupling, code readability, modular architecture.

1. Введение

При разработке программного обеспечения с помощью подходов объектно-ориентированного программирования важно придерживаться общих патернов и принципов проектирования классов и объектов. В данной статье рассмотрим одни из самых главных принципов создания модульной архитектуры приложений: слабую степень связности и высокое зацепление.

Рассмотрим определение модульной связности.

Идею связности впервые представили Уэйн Стивенс, Гленфорд Майерс и Ларри Константайн. На уровне проектирования классов ее практически вытеснили более современные концепции, такие как абстракция и инкапсуляция, однако на уровне проектирования отдельных методов эвристический принцип связности по-прежнему полезен [1].

Связность модуля – его внутренняя характеристика, характеризующая меру прочности соединения функциональных и информационных объектов внутри одного модуля [2].

Существует 7 типов связности: функциональная, последовательная, информационная, процедурная, временная, логическая, случайная.

Функционально-связный модуль содержит объекты, предназначенные для решения одной-единственной задачи.

В последовательно-связном модуле его объекты охватывают подзадачи, для которых выходные данные одной из подзадач являются входными для другой.

Информационно-связный модуль содержит объекты, использующие одни и те же входные или выходные данные.

Объекты процедурно-связного модуля включены в различные подзадачи, в которых управление переходит от одной подзадачи к следующей.

Объекты модуля с временной связностью привязаны к конкретному промежутку времени.

Объекты модуля с логической связностью содействуют решению одной общей подзадачи, для которой они отобраны во внешнем по отношению к модулю мире.

Модуль со случайной связностью содержит объекты, которые слабо связаны друг с другом.

Далее рассмотрим понятие модульного зацепления.

Зацеплением является способ и степень взаимозависимости между программными модулями.

Метрики зацепления и связности были придуманы Ларри Константином, изначальным разработчиком структурного проектирования [3].

Соблюдение слабой степени связности и высокого зацепления обеспечивает:

– независимость и устойчивость классов ко внешним изменениям;

– простоту при повторном использовании кода;

– легкую поддержку кода.

2. Цель исследования

Одной из наиболее сложных задач при разработке модульной архитектуры является поддержка хорошо документированного кода и повышение быстродействия программы. Сложно оценить, каким наилучшим образом создавать классы и взаимодействие между ними в рамках одного модуля.

Цель и задачи исследования. Цель данного исследования – оценить влияние слабой связности и высокого зацепления на быстродействие программы и читаемость кода на примере двух web-приложений, созданных при помощи Rails API и Rails Grape соответственно.

Рассмотрим визуализацию архитектуры программ при помощи диаграммы компонентов [4].

Реализация RailsAPI приложения будет состоять из 4 компонентов в рамках одного модуля, как показано на рис. 1.

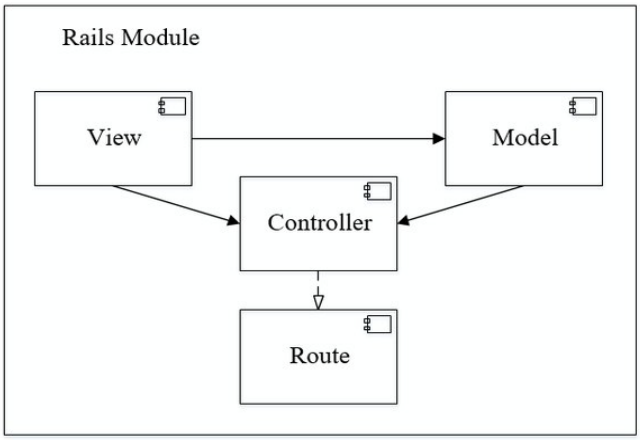

Рис. 1. Компоненты RailsAPI

Реализация GrapeAPI приложения будет состоять из 2 модулей, связанных между собой, как показано на рис. 2.

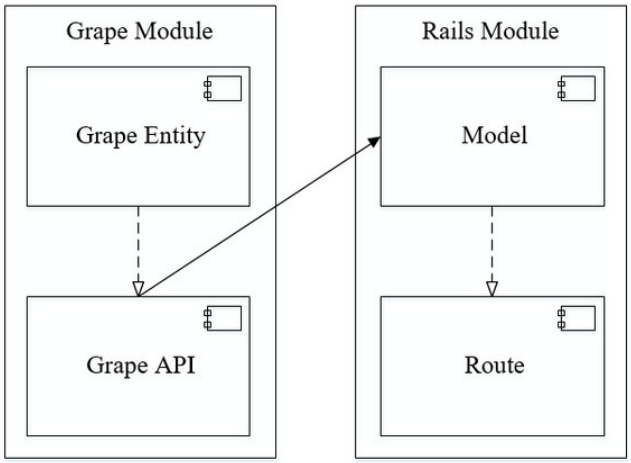

Рис. 2. Компоненты GrapeRailsAPI

Как видно на диаграммах компонентов: RailsAPI приложение имеет один главный модуль, реализующий основной функционал внутри себя; GrapeAPI приложение будет иметь два модуля, которые реализуют свой функционал независимо. Можно сделать вывод, что в GrapeAPI приложении реализовано более сильное зацепление и слабая связность по сравнению с приложением на RailsAPI.

Для достижения поставленной цели необходимо решить следующие задачи:

– для чистоты эксперимента написать два приложения, реализующих одинаковую логику аутентификации пользователей в системе (регистрация, вход, выход);

– при помощи ApacheJMeteor определить быстродействие системы;

- при помощи инструмента flog определить сложность кода.

3. Коды программ

С учетом всего сказанного были созданы два приложения на RailsAPI и GrapeRailsAPI. В них реализона одинаковая логика регистрации, авторизации, входа и выхода из системы для API для протокола OAuth2. Приложения написаны на языке программирования Ruby для более быстрого проведения исследования [5].

Пример кода для входа в систему RailsAPI показан на рис. 3.

```ruby
class Api::V1::SessionsController < Api::V1::BaseController
  skip_before_filter :get_current_user!, only: [:create, :update]

  def create
    result = Authentication::SigninService.new(params).perform

    if result.success?
      @user = result.user
      set_headers(result)
      render :show
    else
      render_error 422, :NotValid, result.render_error
    end
  end
end
```

Рис. 3. RailsAPI вход в систему

Пример кода для входа в систему GrapeRailsAPI показан на рис. 4.

Как видно из примеров программ, код для RailsAPI менее объемный и документированный, похож на чистый Ruby больше, чем код на GrapeAPI.

4. Проведение исследования

Для исследования на быстродействие воспользуемся программным обеспечения ApacheJMeteor, которое позволяет создать нагрузочное тестирование при помощи HTTP-запросов на сервер. Будет проведено 100 измерений на каждый эндпойнт.

Объектом тестирования будет минимальное, максимальное и среднее время ответа сервера, пропускная способность. Для начала запустим сервер на локальной машине. Результаты данного исследования показаны в табл. 1.

Основываясь на полученных данных табл. 1 (среднее время ответа, пропускная способность), можно сделать вывод, что приложение на GrapeAPI рабо-

тает быстрее на всех эндпойнтах по сравнению с RailsAPI.

```ruby
class V1::Sessions < Grape::API
  resources :sessions do
    desc 'User login, create session' do
      success V1::Entities::User
      failure [[401, 'LoginError',
      V1::Entities::Errors::CommonError]]
    end
    params do
      requires :user, type: Hash do
        requires :email, type: String,
                 desc: 'User email'
        requires :password, type: String,
                 desc: 'User password'
      end
      requires :session, type: Hash do
        requires :device_kind, type: String,
                 desc: 'Device kind',
                 values: Session.device_kinds.keys
        requires :push_token, type: String,
                 desc: 'Push token'
      end
      post do
        user = User.find_by!(
          email: declared(params).user.email)
        unless user.authenticate(
          declared(params).user.password)
          raise Exceptions::LoginError
        end
        provider = Providers::Email
                     .find_or_create_by!(user: user)
        session = user.sessions
                  .create!(declared(params)
                  .session.merge(
                    provider_id: provider.id))
        make_headers_for session
        present user,
                with: V1::Entities::User
      end
    end
  end
```

Рис. 4. GrapeRailsAPI вход в систему

Для исследования читабельности кода воспользуемся инструментов flog, который считает количество очков Ruby кода, основываясь на общих паттернах. Чем больше очков набирает файл с кодом, тем более он трудночитабелен, так как возрастает количество вызовов методов и классов и созданных переменных.

Объектом исследования являются файлы, в которых реализован функционал приложения. Результаты данного исследования показаны в табл. 2.

Таблица 1

Название	Мин. время, мс	Ср. время, мс	Макс. время, мс	Проп. спос., кол./с
Регистрация, Rails API	187	860	1582	5.6
Регистрация, Grape API	219	741	1415	6.5
Вход, Rails API	320	819	1125	5.9
Вход, Grape API	138	628	884	7.6
Выход, Rails API	7	13	33	94.2
Выход, Grape API	3	6	179	111.6

Таблица 2

Название	Общая сложность	Средняя сложность
Регистрация, Rails API	186.6	5.2
Регистрация, Grape API	64.3	10.7
Вход, Rails API	233.3	5.7
Вход, Grape API	133.6	13.4
Выход, Rails API	64.9	7.2
Выход, Grape API	122.7	13.6

Основываясь на полученных данных табл. 2, можно сделать вывод, что внедрение разработчика в проект с приложением на RailsAPI быстрее и легче, так как средняя сложность Ruby кода примерно в 2 раза меньше, чем в GrapeAPI приложении.

5. Выводы

Научная новизна. Выявлено влияние принципов проектирования (слабая связность и сильное зацепление) на быстродействие программы и читабельности кода.

Практическая значимость. Выявлено, что уменьшение связности модулей и увеличение зацепления ускоряет быстродействие системы, одновременно увеличивая сложность кода, на примере

сравнения приложений, созданных разными способами, в среде Rails разработки.

Учитывая все сказанное выше, можно сделать вывод, что для разработки проектов, требующих гибкой и качественной реализации с возможностью последующих изменений без затрагивания всего функционала, важно применять основные принципы проектирования архитектуры. Среднее значение сложности кода будет однозначно выше, чем при пренебрежении паттернов, однако это сохранит время на внедрение новых разработчиков в проект и последующие изменения кода, а также может увеличить быстродействие системы.

Литература: **1.***Макконнелл С.* Совершенный код. М. Русская редакция, 2010. 896 с. **2.***Связность* (программирование): электронный ресурс. Точка доступа https://en.wikipedia.org/wiki/Cohesion. **3.** *Константайн Л., Майерс Г., Стивенс У.* Структурированный дизайн // IBMSystemsJournal. 1974. Т. 13, № 2. С. 115—139. **4.***Ларман К.* Применение UML и шаблонов проектирования. М. Вильямс, 2002. 624 с. **5.** *Фитцджеральд М.* Изучаем Ruby та Ruby on Rails на примере приложения. Санкт-Петербург.: БХВ-Петербург, 2008. 336 с.

Transliterated bibliography:

1. *Makkonnell S.* Sovershennyjkod. Moskva: Russkajaredakcija, 2010.896 s.

2. *Svjaznost' (programmirovanie):* jelektronnyjresurs. Tochkadostupa https://en.wikipedia.org/wiki/Cohesion.

3. *Konstantajn L., Majers G., Stivens U.* Strukturirovannyjdizajn // IBM Systems Journal. 1974. T. 13, № 2. S. 115–139.

4. *Larman K.* Primenenie UML ishablonovproektirovanija. Moskva: Vil'jams, 2002. 624 s.

5. *Fitcdzheral'd M.* Izuchaem Ruby ta Ruby on Rails naprimereprilozhenija. Sankt-Peterburg.BHV-Peterburg, 2008. 336 s.

Поступила в редколлегию 11.09.2016

Рецензент: д-р техн.наук, проф. Ерохин А.Л.

Кравченко Александр Константинович, студент кафедры ПИ ХНУРЭ. Научные интересы: информационные технологии. Адрес: Украина, 61166, Харьков, пр. Науки, 14.Email: oleksandr.kravchenko@nure.ua.

Афанасьева Ирина Витальевна, канд. техн.наук, доцент кафедры ПИ ХНУРЭ. Научные интересы: информационные технологии, искуственный интеллект. Адрес: Украина, 61166, Харьков, пр. Науки, 14. Email: irina.kamenieva@gmail.com.

Kravchenko Alexander Konstantinovich, student of Program Engineering of the Kharkov National University of Radio Electronics. Scientific interests: information technology. Address: Ukraine, 61166, Kharkov, 14, Nauki Avenue. Email: oleksandr.kravchenko@nure.ua.

Afanasyeva Irina Vitalevna, Ph.D., Assistant Professor of Program Engineering of the Kharkov National University of Radio Electronics. Scientific interests: information technology, artificial intelligence. Address: Ukraine, 61166, Kharkov, 14, Nauki Ave. Email: irina.kamenieva@gmail.com.