Для обнаружения зерен был использован алгоритм «FloodFill» [4], визуальная интерпретация которого представлена на рисунке 6.

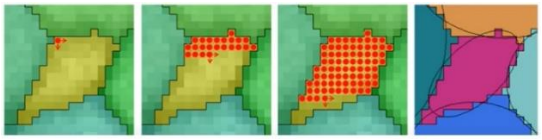

Рис. 6. Алгоритм «FloodFill»

В результате проведения данной работы была разработана информационная система для анализа микроструктуры металлов на основе данных, полученных методом ДОЭ, изучен фреймворк для работы с вычислительными мощностями видеокарты, а также методы очистки данных ДОЭ, определения границ, картирования ориентировок и т.п. Изучено несколько алгоритмов, используемых при работе с изображениями и применены для решения задач анализа. Было разработано несколько вариантов пользовательского интерфейса. Система была протестирована на различных вариантах исходных данных.

Список использованных источников

1. Варюхин В.Н., Пашинская Е.Г., Завдовеев А.В., Бруховецкий В.В. Возможности метода дифракции обратнорассеяных электронов для анализа структуры деформированных материалов. – Национальная академия наук Украины, Донецкий физико-технический институт им. А.А. Галкина, 2014. – 101 с.
2. Open standard for parallel programming of heterogeneous systems [Электронный ресурс] – URL: https://www.khronos.org/opencl/ (дата обращения: 10.10.2021).
3. Kuwahara filter [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/Kuwahara_filter (дата обращения: 10.10.2021).
4. Flood fill [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/Flood_fill (дата обращения: 10.10.2021).

УДК 004.273

А. А. Ершов, М. Ю. Ляшенко, М. Д. Бурцев
ФГБОУ ВО «Магнитогорский государственный технический университет им. Г.И. Носова», г. Магнитогорск, Россия

ЧИСТАЯ АРХИТЕКТУРА ПО НА PHP

Аннотация. *В статье приведены основные принципы архитектуры ПО на примере языка PHP, а также их применение в проектах, взаимосвязь отдельных модулей друг с другом и их индивидуальное значение, функционал и результат их деятельности, согласно ар-*

хитектуре языка PHP.Раскрывается понятие “Чистой архитектуры” кода и как она влияет на разработку проектов, её основные принципы и правила

Ключевые слова: *Архитектура PHP, чистая архитектура, программный код, модули, приложение, проект, MVC, SOLID.*

Abstract. *The article presents the basic principles of the software architecture on the example of the PHP language, as well as their application in projects, the relationship of individual modules with each other and their individual meaning, functionality and result of their activities, according to the architecture of the PHP language. The concept of “Clean architecture” of the code is revealed and how it affects project development, its basic principles and rules.*

Key words: *PHP architecture, clean architecture, code, modules, application, project, MVC, SOLID.*

Архитектура ПО – совокупность важнейших решений об организации программной системы.

Архитектура включает:

1. Выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов;

2. Соединение выбранных элементов структуры и поведения во всё более крупные системы;

3. Архитектурный стиль, который направляет всю организацию – все элементы, их интерфейсы, их сотрудничество и их соединение.

Документирование архитектуры программного обеспечения (ПО) упрощает процесс коммуникации между разработчиками, позволяет зафиксировать принятые проектные решения и предоставить информацию о них эксплуатационному персоналу системы, повторно использовать компоненты и шаблоны проекта в других. Основополагающей идеей дисциплины программной архитектуры является идея снижения сложности системы путём абстракции и разграничения полномочий. На сегодняшний день до сих пор нет согласия в отношении чёткого определения термина «архитектура программного обеспечения».

Являясь в настоящий момент своего развития дисциплиной без четких правил о «правильном» пути создания системы, проектирование архитектуры ПО все ещё является смесью науки и искусства. Аспект «искусства» заключается в том, что любая коммерческая система подразумевает наличие применения или миссии. С точки зрения пользователя программной архитектуры, программная архитектура дает направление для движения и решения задач, связанных со специальностью каждого такого пользователя, например, заинтересованного лица, разработчика ПО, группы поддержки ПО, специалиста по сопровождению ПО, специалиста по развертыванию ПО, тестера, а также конечных пользователей. В этом смысле архитектура программного обеспечения на самом деле объединяет различные точки зрения на систему. Тот факт, что эти несколько различных точек зрения могут быть объединены в архитектуре программного обеспечения, является аргументом в защиту необходимости и целесообразности создания архитектуры ПО ещё до этапа разработки ПО. Архитектура ПО обычно содержит несколько видов, которые аналогичны различным

типам чертежей в строительстве зданий. Архитектурный вид состоит из двух компонентов:

1. Элементы.

2. Отношения между элементами.

Архитектурные виды можно поделить на три основных типа:

1. Модульные виды – показывают систему как структуру из различных программных блоков.

2. Компоненты-и-коннекторы – показывают систему как структуру из параллельно запущенных элементов (компонентов) и способов их взаимодействия (коннекторов).

3. Размещение – показывает размещение элементов системы во внешних средах.

Для удовлетворения проектируемой системы различным атрибутам качества применяются различные архитектурные шаблоны (паттерны). Каждый шаблон имеет свои задачи и свои недостатки.

Примеры архитектурных шаблонов:

1. Многоуровневый шаблон. Система разбивается на уровни, которые на диаграмме изображаются один над другим. Каждый уровень может вызывать только уровень на 1 ниже него. Таким образом, разработку каждого уровня можно вести относительно независимо, что повышает модифицируемость системы. Недостатками данного подхода являются усложнение системы и снижение производительности.

2. Шаблон посредника. Когда в системе присутствует большое количество модулей, их прямое взаимодействие друг с другом становится слишком сложным. Для решения проблемы вводится посредник (например, шина данных), по которой модули общаются друг с другом. Таким образом, повышается функциональная совместимость модулей системы. Все недостатки вытекают из наличия посредника: он понижает производительность, его недоступность может сделать недоступной всю систему, он может стать объектом атак и узким местом системы.

3. Шаблон «Модель-Представление-Контроллер» (рис. 1). Поскольку требования к интерфейсу меняются чаще всего, то возникает потребность часто его модифицировать, при этом сохраняя корректное взаимодействие с данными (чтение, сохранение). Для этого в шаблоне Model-View-Controller (MVC) интерфейс отделён от данных. Это позволяет менять интерфейсы, равно как и создавать их разные варианты. В MVC система разделена на:

– модель, хранящую данные;

– представление, отображающее часть данных и взаимодействующее с пользователем;

– контроллер, являющийся посредником между видами и моделью.

Однако концепция MVC имеет и свои недостатки. В частности, из-за усложнения взаимодействия падает скорость работы системы.

Шаблон MVC рассмотрим детально, так как он отлично подходит под создание приложений на языке PHP, ведь архитектура приложения должна отражать в себе концепцию того, как вообще работает приложение. Если говорить о

проектах на языке PHP, то в подавляющем большинстве случаев – это веб-сайты. Согласитесь, все веб-сайты работают примерно одинаково:

1. Получение и обработка запроса от пользователя (GET-запрос на страничку со статьёй).
2. Понимание того, как на этот запрос нужно отреагировать (получить статью из базы данных и вернуть её пользователю).
3. Работа с данными, их получение/изменение в базе данных (получение статьи из базы данных).
4. Формирование представления для пользователя (заполнение HTML-шаблона данными из базы данных).
5. Отправка ответа пользователю (отправка сформированной HTML-странички с текстом статьи).

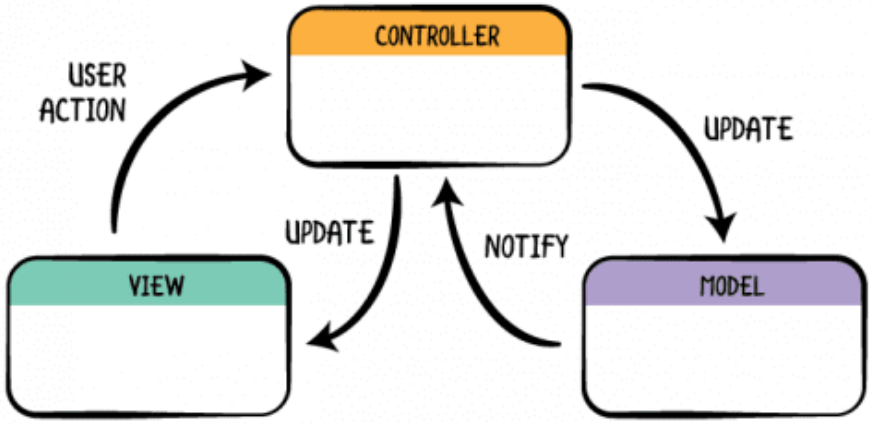

Рис. 1. Модель шаблона MVC

Модель – это часть приложения, работающая с данными. Она содержит в себе данные и умеет их отображать в базу данных. То есть она может добавлять записи в базу данных, удалять их, изменять, или же просто получать их оттуда. Задача модели – взять данные и передать их тому, кто эти данные у неё запрашивает. Если мы посмотрим на рисунок того, как устроена архитектура MVC, мы видим, что модель взаимодействует с контроллером. Таким образом, контроллер может получать данные от модели, либо же передавать эти данные в модель. С другой стороны от модели, как правило, находится база данных, в которой модель эти данные умеет хранить и получать их оттуда.

View (представление) – Эта часть программы довольно проста и отвечает за то, в каком виде пользователь получит данные от приложения. В этот блок приходят какие-то данные от контроллера, например, имя HTML-шаблона и переменные, которые в этот шаблон нужно передать.

Controller (контроллер) – это связующее звено между запросом от пользователя, моделями и представлением. Именно контроллер является точкой входа в приложение. Сюда приходит запрос от пользователя и принимается решение о том, что с этим запросом делать. Например, создать новую статью. Тогда контроллер создаёт новую модель данных для статьи и просит её сохраниться в базу данных. Затем он берёт эту статью и передаёт её в представление. Представление берёт шаблон для вывода статьи и переменные, полученные от контрол-

лера (заголовок, текст статьи, её автора) и после этого возвращает сформированную страничку пользователю.

Понятие «архитектура чистого кода» (Clean Code Architecture) ввел Роберт Мартин в блоге 8light (рис. 2). Смысл понятия в том, чтобы создавать архитектуру, которая не зависела бы от внешнего воздействия. Ваша бизнес-логика не должна быть объединена с фреймворком, базой данных или самим вебом. Подобная независимость даёт ряд преимуществ.

Рис. 2. Модель архитектуры с чистым кодом

К примеру, при разработке вы сможете откладывать какие-то технические решения, например выбор фреймворка, движка/поставщика БД. Также вы сможете легко переключаться между разными реализациями и сравнивать их. Но самое важное преимущество такого подхода — ваши тесты будут выполняться быстрее.

На этой иллюстрации изображены разные слои приложения. Внутренние ничего не знают о внешних, при этом все они взаимодействуют друг с другом через интерфейсы.

Самое интересное – в правом нижнем углу: поток управления. Схема объясняет, как фреймворк взаимодействует с бизнес-логикой. Контроллер передаёт данные на порт ввода, информацию с которого обрабатывает интерактор, а результат передаётся на порт вывода, содержащий данные для презентера.

Список использованных источников

1. Архитектура чистого кода и разработка через тестирование в PHP – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/vk/blog/277543/.

2. Архитектура программного обеспечения– [Электронный ресурс]. – Режим доступа:

https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F.

3. Чистая архитектура на PHP. Как её измерять и контролировать? – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/504590/.

4. Архитектура MVC – [Электронный ресурс]. – Режим доступа: https://php.zone/oop-v-php-prodvinutyj-kurs/arhitektura-prilozheniya-i-pattern-mvc.

УДК: 681.5:669.1:628.511

А. К. Ершов, В. А. Гольцев
ФГАОУ ВО «Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина», г. Екатеринбург, Россия

РАЗРАБОТКА СИСТЕМЫ АВТОМАТИЗАЦИИ ПЫЛЕУДАЛЕНИЯ ИЗ РАБОЧЕГО ПРОСТРАНСТВА ЦЕХА

Аннотация. *Выявлено, что в цехе присутствуют как крупнодисперсные, так и мелкодисперсные частицы марганца и диоксида кремния, основными источниками которых являются руднотермические печи. В состав пыли в рабочем пространстве цеха вошли диоксид кремния, марганец и его соединения, а также оксиды кальция и алюминия, которые являются очень вредоносными для легочной системы работающего в цеху персонала. Разработана система автоматизированной пыле очистки на базе программируемого реле ПР200 и датчиков пыли AirSafe2. Разработана и система сигнализации, информирующая персонал о запыленности и аварии оборудования, и вывод информации о всей системе в SCADA систему предприятия.*

Ключевые слова: *силикомарганец, диоксид кремния, руднотермическая печь, автоматизация, контроллер, пыль.*

Abstract. With the help of special devices for analyzing the concentration and composition of dust, the aerodisperse system of the workshop was analyzed. It was revealed that in the workshop there are both coarse and fine particles of manganese and silicon dioxide, the main sources of which are ore-thermal furnaces. The composition of the dust in the working space of the workshop included silicon dioxide, manganese and its compounds, as well as calcium and aluminum oxides, which are very harmful to the pulmonary system of the personnel working in the workshop. An automated dust cleaning system based on the PR200 programmable relay and AirSafe2 dust sensors has been developed. An alarm system has also been developed that informs personnel about dustiness and equipment failures, and the output of information about the entire system to the SCADA system of the enterprise.

Key words: *silicomanganese, silicon dioxide, ore-thermal furnace, automation, controller, dust.*

Объектом разработки системы автоматизации послужил плавильный цех с установленными руднотермическими печами открытого типа РКО-23 с угольной футеровкой и тремя самоспекающимися электродами в количестве 4 штук, и 4 закрытых печей типа РКЗ-23, предназначенных для выплавки кремнистых и