Глава 2: Дизайн программного обеспечения

Из SWEBOK

Содержание

АКРОНИМЫ

АДЛ	Язык описания архитектуры
КБР	Компонентный дизайн
CRC	Ответственность за класс Сотрудник
ДФД	Диаграмма потока данных
ЭРД	Диаграмма отношений сущностей
IDL	Язык описания интерфейса
МВК	Контроллер представления модели
ООО	Объектно-ориентированный
ПДЛ	Язык разработки программы

ВВЕДЕНИЕ

Дизайн определяется как «процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или компонента», так и «результат [этого] процесса» [1]. Рассматриваемый как процесс, проектирование программного обеспечения представляет собой деятельность жизненного цикла разработки программного обеспечения, в ходе которой анализируются требования к программному обеспечению для получения описания внутренней структуры программного обеспечения, которое послужит основой для его построения. Проект программного обеспечения (результат) описывает архитектуру программного обеспечения, то есть то, как программное обеспечение разлагается и организуется на компоненты, а также интерфейсы между этими компонентами. Он также должен описывать компоненты на таком уровне детализации, который позволяет их построение.

Дизайн программного обеспечения играет важную роль в разработке программного обеспечения: во время проектирования программного обеспечения инженеры-программисты создают различные модели, которые формируют своего рода план решения, которое необходимо реализовать. Мы можем проанализировать и оценить эти модели, чтобы определить, позволят ли они нам выполнить различные требования.

Мы также можем изучить и оценить альтернативные решения и компромиссы. Наконец, мы можем использовать полученные модели для планирования последующих действий по разработке, таких как проверка и проверка системы, в дополнение к их использованию в качестве входных данных и в качестве отправной точки для построения и тестирования. В стандартном списке процессов жизненного цикла программного обеспечения, например, в ISO/IEC/IEEE Std. 12207, Процессы жизненного цикла программного обеспечения [2], проектирование программного обеспечения состоит из двух действий, которые соответствуют анализу требований к программному обеспечению и созданию программного обеспечения:

- Проект архитектуры программного обеспечения (иногда называемый проектированием высокого уровня): разрабатывает структуру и организацию программного обеспечения верхнего уровня и определяет различные компоненты.
- Детальный проект программного обеспечения: определяет каждый компонент достаточно подробно, чтобы облегчить его создание.

В этой области знаний по дизайну программного обеспечения (KA) не обсуждаются все темы, в которых есть слово «дизайн». В терминологии Тома Марко [3] темы, обсуждаемые в этом KA, касаются в основном D-дизайна (дизайна декомпозиции), целью которого является отображение программного

обеспечения на составные части. Однако из-за его важности в области архитектуры программного обеспечения мы также обратимся к FP-дизайну (дизайну семейства паттернов), целью которого является установление пригодных для использования общих черт в семействе программных продуктов. Этот KA не рассматривает I-дизайн (проектирование изобретения), который обычно выполняется в процессе разработки требований к программному обеспечению с целью концептуализации и спецификации программного обеспечения для удовлетворения обнаруженных потребностей и требований, поскольку эта тема считается частью процесса требований (см. Требования к программному обеспечению KA).

Этот KA «Проектирование программного обеспечения» конкретно связан с KA «Требования к программному обеспечению», «Создание программного обеспечения», «Управление разработкой программного обеспечения», «Модели и методы разработки программного обеспечения», «Качество программного обеспечения» и «Основы вычислений».

РАЗБИВКА ТЕМ ДЛЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Разбивка тем для KA «Проектирование программного обеспечения» показана на рисунке 2.1.

1 Основы проектирования программного обеспечения

Введенные здесь концепции, понятия и терминология составляют основу для понимания роли и области проектирования программного обеспечения.

1.1 Общие концепции дизайна

[4 , c1]

В общем смысле дизайн можно рассматривать как форму решения проблем. Например, концепция коварной проблемы — проблемы, не имеющей окончательного решения, — интересна с точки зрения понимания пределов замысла. Ряд других понятий и концепций также представляет интерес для понимания дизайна в его общем смысле: цели, ограничения, альтернативы, представления и решения (см. Методы решения проблем в Computing Foundations KA).

1.2 Контекст разработки программного обеспечения

[4 , c3]

Проектирование программного обеспечения является важной частью процесса разработки программного обеспечения. Чтобы понять роль дизайна программного обеспечения, мы должны увидеть, как он вписывается в жизненный цикл разработки программного обеспечения. Таким образом, важно понимать основные характеристики анализа требований к программному обеспечению, проектирования программного обеспечения, построения программного обеспечения, тестирования программного обеспечения и обслуживания программного обеспечения.

1.3 Процесс разработки программного обеспечения

[4 , c2]

Разработка программного обеспечения обычно считается двухэтапным процессом:

- Архитектурный дизайн (также называемый дизайном высокого уровня и дизайном верхнего уровня) описывает, как программное обеспечение организовано в компоненты.
- Детальный проект описывает желаемое поведение этих компонентов.

Результатом этих двух процессов является набор моделей и артефактов, которые фиксируют основные принятые решения, а также поясняют обоснование каждого нетривиального решения. Записывая обоснование, улучшается долгосрочная ремонтопригодность программного продукта.

1.4 Принципы проектирования программного обеспечения

[4] [5 , c6, c7, c21] [6 , c1, c8, c9]

Принцип – это «всеобъемлющий и фундаментальный закон, доктрина или предположение» [7] . Принципы проектирования программного обеспечения являются ключевыми понятиями, которые обеспечивают основу для многих различных подходов и концепций проектирования программного

обеспечения. Принципы проектирования программного обеспечения включают абстракцию; сцепление и сплоченность; декомпозиция и модуляризация; инкапсуляция/сокрытие информации; разделение интерфейса и реализации; достаточность, полнота и примитивность; и разделение интересов.

- *Абстракция* — это «представление объекта, которое сосредотачивается на информации, относящейся к конкретной цели, и игнорирует остальную часть информации» [1] (см. Абстракция в Computing Foundations KA). В контексте проектирования программного обеспечения двумя ключевыми механизмами абстракции являются параметризация и спецификация. Абстракция посредством параметризации абстрагируется от деталей представления данных, представляя данные в виде именованных параметров. Абстракция спецификацией приводит к трем основным видам абстракции: процедурной абстракции, абстракции данных и абстракции управления (итерации).
- *Сцепление и сплоченность* . Связь определяется как «мера взаимозависимости между модулями в компьютерной программе», тогда как связность определяется как «мера силы ассоциации элементов внутри модуля» [1].
- *Декомпозиция и модуляризация* . Декомпозиция и модульность означают, что большое программное обеспечение разделено на ряд более мелких именованных компонентов, имеющих четко определенные интерфейсы, описывающие взаимодействие компонентов. Обычно цель состоит в том, чтобы разместить разные функции и обязанности в разных компонентах.
- *Инкапсуляция и сокрытие информации* означают группировку и упаковку внутренних деталей абстракции и обеспечение недоступности этих деталей для внешних сущностей.
- *Разделение интерфейса и реализации* . Разделение интерфейса и реализации включает в себя определение компонента путем указания общедоступного интерфейса (известного клиентам), который отделен от деталей реализации компонента (см. инкапсуляцию и сокрытие информации выше).
- *Достаточность, полнота и примитивность* . Достижение достаточности и полноты означает обеспечение того, чтобы программный компонент охватывал все важные характеристики абстракции и ничего более. Примитивность означает, что дизайн должен быть основан на шаблонах, которые легко реализовать.
- *Разделение забот* . Проблема — это «область интереса в отношении дизайна программного обеспечения» [8]. Проблема дизайна — это область дизайна, которая имеет отношение к одному или нескольким заинтересованным сторонам. Каждое архитектурное представление обрамляет одну или несколько задач. Разделение проблем по представлениям позволяет заинтересованным сторонам сосредоточиться на нескольких вещах одновременно и предлагает средства управления сложностью [9].

2 ключевых вопроса разработки программного обеспечения

При разработке программного обеспечения необходимо решить ряд ключевых вопросов. Некоторые из них относятся к проблемам качества, которые должно решать все программное обеспечение, например, производительность, безопасность, надежность, удобство использования и т. д. Другой важный вопрос заключается в том, как декомпозировать, организовывать и упаковывать программные компоненты. Это настолько фундаментально, что все подходы к проектированию так или иначе учитывают это (см. раздел 1.4 «Принципы проектирования программного обеспечения» и тему 7 «Стратегии и методы проектирования программного обеспечения»). В отличие от этого, другие проблемы «имеют дело с некоторыми аспектами поведения программного обеспечения, которые не относятся к предметной области, но относятся к некоторым поддерживающим областям» [10]. Такие проблемы, которые часто пересекаются с функциональностью системы, называются аспектами, которые «как правило, не являются единицами функциональной декомпозиции программного обеспечения, а скорее быть свойствами, которые системно влияют на производительность или семантику компонентов» [11]. Некоторые из этих ключевых сквозных вопросов обсуждаются в следующих разделах (представлены в алфавитном порядке).

2.1 Параллелизм

[5 , c18]

Проектирование для параллелизма связано с декомпозицией программного обеспечения на процессы, задачи и потоки и решением связанных вопросов эффективности, атомарности, синхронизации и планирования.

2.2 Контроль и обработка событий

[5 , c21]

Эта проблема проектирования связана с тем, как организовать данные и поток управления, а также как обрабатывать реактивные и временные события с помощью различных механизмов, таких как неявный вызов и обратные вызовы.

2.3 Постоянство данных

[12 , c9]

Эта проблема проектирования связана с тем, как обрабатывать долгоживущие данные.

2.4 Распределение компонентов

[5 , c18]

Этот вопрос проектирования связан с тем, как распределять программное обеспечение по оборудованию (включая компьютерное оборудование и сетевое оборудование), как взаимодействуют компоненты и как промежуточное программное обеспечение может использоваться для работы с гетерогенным программным обеспечением.

2.5 Обработка ошибок и исключений и отказоустойчивость

[5 , c18]

Этот вопрос проектирования связан с тем, как предотвращать, допускать и обрабатывать ошибки, а также справляться с исключительными условиями.

2.6 Взаимодействие и представление

[5 , c16]

Этот вопрос дизайна касается того, как структурировать и организовать взаимодействие с пользователями, а также представление информации (например, разделение представления и бизнес-логики с использованием подхода Модель-Представление-Контроллер). Обратите внимание, что в этом разделе не указываются детали пользовательского интерфейса, что является задачей проектирования пользовательского интерфейса (см. раздел 4, Проектирование пользовательского интерфейса).

2.7 Безопасность

[5 , c12, c18] [3 , c4]

Дизайн для обеспечения безопасности связан с тем, как предотвратить несанкционированное раскрытие, создание, изменение, удаление или отказ в доступе к информации и другим ресурсам. Он также касается того, как справляться с атаками или нарушениями, связанными с безопасностью, путем ограничения ущерба, продолжения обслуживания, ускорения ремонта и восстановления, а также безопасного сбоя и восстановления. Контроль доступа является фундаментальной концепцией безопасности, и необходимо также обеспечить правильное использование криптологии.

3 Структура и архитектура программного обеспечения

Строго говоря, программная архитектура — это «набор структур, необходимых для рассуждений о системе, которые включают программные элементы, отношения между ними и их свойства» [14*]. Однако в середине 1990-х архитектура программного обеспечения начала формироваться как более широкая дисциплина, включающая изучение структур и архитектур программного обеспечения в более общем виде. Это породило ряд интересных концепций проектирования программного обеспечения на разных уровнях абстракции. Некоторые из этих концепций могут быть полезны как при архитектурном проектировании (например, архитектурные стили), так и при детальном проектировании (например, шаблоны проектирования). Эти концепции дизайна также можно использовать для разработки семейств программ (также известных как линейки продуктов). Интересно,

3.1 Архитектурные конструкции и точки зрения

[14 , c1]

Можно описать и задокументировать различные высокоуровневые аспекты разработки программного обеспечения. Эти аспекты часто называют представлениями: «Представление представляет собой частичный аспект программной архитектуры, который показывает определенные свойства программной системы» [14*]. Представления относятся к различным проблемам, связанным с проектированием программного обеспечения, например, логическое представление (удовлетворение функциональных требований) и представление процесса (вопросы параллелизма), физическое представление (проблемы распространения) и представление разработки (как разработан проект). разбиты на блоки реализации с явным представлением зависимостей между блоками). Различные авторы используют разные термины, например, поведенческие, функциональные, структурные и представления моделирования данных. В итоге,

3.2 Архитектурные стили

[14 , c1, c2, c3, c4, c5]

Архитектурный стиль — это «специализация типов элементов и отношений вместе с набором ограничений на то, как их можно использовать» [14*]. Таким образом, архитектурный стиль можно рассматривать как обеспечивающий высокоуровневую организацию программного обеспечения. Различные авторы выделяют ряд основных архитектурных стилей:

- Общие структуры (например, слои, трубы и фильтры, классная доска)
- Распределенные системы (например, клиент-серверные, трехуровневые, брокерские)
- Интерактивные системы (например, Model-View-Controller, Presentation-Abstraction-Control)
- Адаптируемые системы (например, микроядро, рефлексия)
- Другие (например, пакетные, интерпретаторы, управление процессами, на основе правил).

3.3 Шаблоны проектирования

[15 , c3, c4, c5]

В кратком описании паттерн — это «обычное решение общей проблемы в заданном контексте» [16]. В то время как архитектурные стили можно рассматривать как шаблоны, описывающие высокоуровневую организацию программного обеспечения, другие шаблоны проектирования можно использовать для описания деталей на более низком уровне. Эти шаблоны проектирования более низкого уровня включают следующее:

- Шаблоны создания (например, конструктор, фабрика, прототип, синглтон)
- Структурные паттерны (например, адаптер, мост, составной элемент, декоратор, фасад, легковес, прокси)
- Поведенческие паттерны (например, команда, интерпретатор, итератор, посредник, сувенир, наблюдатель, состояние, стратегия, шаблон, посетитель).

3.4 Решения по проектированию архитектуры

[5 , c6]

Архитектурное проектирование – это творческий процесс. В процессе проектирования разработчики программного обеспечения должны принять ряд фундаментальных решений, которые глубоко влияют на программное обеспечение и процесс разработки. Полезно рассматривать процесс архитектурного проектирования с точки зрения принятия решений, а не с точки зрения деятельности. Часто влияние на атрибуты качества и компромиссы между конкурирующими атрибутами качества являются основой для проектных решений.

3.5 Семейства программ и фреймворков

[5 , c6, c7, c16]

Один из подходов к обеспечению повторного использования проектов и компонентов программного обеспечения заключается в разработке семейств программ, также известных как линейки программных продуктов. Это можно сделать, выявив общие черты среди членов таких семейств и разработав многоразовые и настраиваемые компоненты для учета изменчивости среди членов семейства. В

объектно-ориентированном (ОО) программировании ключевое понятие — это структура: частично завершенная программная система, которая может быть расширена путем создания соответствующих экземпляров конкретных расширений (таких как плагины).

4 Дизайн пользовательского интерфейса

Дизайн пользовательского интерфейса является неотъемлемой частью процесса разработки программного обеспечения. Дизайн пользовательского интерфейса должен гарантировать, что взаимодействие между человеком и машиной обеспечивает эффективную работу и управление машиной. Чтобы программное обеспечение полностью реализовало свой потенциал, пользовательский интерфейс должен быть спроектирован так, чтобы соответствовать навыкам, опыту и ожиданиям предполагаемых пользователей.

4.1 Общие принципы проектирования пользовательского интерфейса

[5 , c29-сетка] [17 , c2]

- *Обучаемость* . Программное обеспечение должно быть простым в освоении, чтобы пользователь мог быстро приступить к работе с ним.
- *Знакомство с пользователем* . В интерфейсе должны использоваться термины и концепции, основанные на опыте людей, которые будут использовать программное обеспечение.
- *Консистенция* . Интерфейс должен быть последовательным, чтобы сопоставимые операции активировались одинаково.
- *Минимальный сюрприз* . Поведение программного обеспечения не должно удивлять пользователей.
- *Восстанавливаемость* . Интерфейс должен предоставлять механизмы, позволяющие пользователям восстанавливаться после ошибок.
- *Руководство пользователя* . Интерфейс должен давать значимую обратную связь при возникновении ошибок и предоставлять контекстную помощь пользователям.
- *Разнообразие пользователей* . Интерфейс должен обеспечивать соответствующие механизмы взаимодействия для различных типов пользователей и пользователей с различными возможностями (слепые, слабовидящие, глухие, дальтоники и т. д.).

4.2 Проблемы проектирования пользовательского интерфейса

[5 , c29-сетка] [17 , c2]

Дизайн пользовательского интерфейса должен решать две ключевые проблемы:

- Как пользователь должен взаимодействовать с программным обеспечением?
- Как информация из программного обеспечения должна быть представлена пользователю?

Дизайн пользовательского интерфейса должен интегрировать взаимодействие с пользователем и представление информации. Дизайн пользовательского интерфейса должен предусматривать компромисс между наиболее подходящими стилями взаимодействия и представления программного обеспечения, опытом и опытом пользователей программного обеспечения и доступными устройствами.

4.3 Дизайн модальностей взаимодействия с пользователем

[5 , c29-сетка] [17 , c2]

Взаимодействие с пользователем включает в себя выдачу команд и предоставление соответствующих данных программному обеспечению. Стили взаимодействия с пользователем можно разделить на следующие основные стили:

- *Вопрос-ответ* . Взаимодействие по существу ограничивается одним обменом вопросами и ответами между пользователем и программным обеспечением. Пользователь задает вопрос программному обеспечению, и программное обеспечение возвращает ответ на вопрос.
- *Прямая манипуляция* . Пользователи взаимодействуют с объектами на экране компьютера. Прямая манипуляция часто включает в себя указывающее устройство (такое как мышь, шаровой манипулятор или палец на сенсорных экранах), которое манипулирует объектом и вызывает действия, определяющие, что нужно делать с этим объектом.
- *Выбор меню* . Пользователь выбирает команду из списка меню команд.

- *Заполнение формы* . Пользователь заполняет поля формы. Иногда поля включают в себя меню, и в этом случае форма имеет кнопки действий, с помощью которых пользователь может инициировать действие.
- *Язык команд* . Пользователь выдает команду и предоставляет соответствующие параметры, чтобы указать программному обеспечению, что делать.
- *Естественный язык* . Пользователь выдает команду на естественном языке. То есть естественный язык является внешним интерфейсом к командному языку, анализируется и транслируется в программные команды.

4.4 Дизайн представления информации

[5 , с29-сетка] [17 , с2]

Представление информации может быть текстовым или графическим. Хороший дизайн отделяет представление информации от самой информации. Подход MVC (Model-View-Controller) — эффективный способ отделить представление информации от представляемой информации. Дизайн программного обеспечения Инженеры-программисты также учитывают время отклика программного обеспечения и обратную связь при разработке представления информации. Время отклика обычно измеряется с момента, когда пользователь выполняет определенное управляющее действие, до момента, когда программное обеспечение отвечает ответом. Желательна индикация прогресса, пока программное обеспечение готовит ответ. Обратная связь может быть предоставлена путем повторного ввода данных пользователем во время завершения обработки. Абстрактные визуализации можно использовать, когда необходимо представить большие объемы информации. В соответствии со стилем подачи информации дизайнеры также могут использовать цвет для улучшения интерфейса. Есть несколько важных рекомендаций:

- Ограничьте количество используемых цветов.
- Используйте изменение цвета, чтобы показать изменение состояния программного обеспечения.
- Используйте цветовое кодирование для поддержки задачи пользователя.
- Используйте цветовое кодирование продуманно и последовательно.
- Используйте цвета, чтобы облегчить доступ для людей с дальтонизмом или цветовой недостаточностью (например, используйте изменение насыщенности цвета и яркости цвета, старайтесь избегать сочетаний синего и красного).
- Не полагайтесь только на цвет, чтобы донести важную информацию до пользователей с различными возможностями (слепыми, плохим зрением, дальтонизмом и т. д.).

4.5 Процесс проектирования пользовательского интерфейса

[5 , с29-сетка] [17 , с2]

Дизайн пользовательского интерфейса — это повторяющийся процесс; прототипы интерфейса часто используются для определения функций, организации и внешнего вида пользовательского интерфейса программного обеспечения. Этот процесс включает в себя три основных действия:

- *Анализ пользователей* . На этом этапе дизайнер анализирует задачи пользователей, рабочую среду, другое программное обеспечение и то, как пользователи взаимодействуют с другими людьми.
- *Программное прототипирование* . Разработка прототипа программного обеспечения помогает пользователям направлять эволюцию интерфейса.
- *Оценка интерфейса* . Дизайнеры могут наблюдать за работой пользователей с развивающимся интерфейсом.

4.6 Локализация и интернационализация

[17 , с8. с9]

Дизайн пользовательского интерфейса часто должен учитывать интернационализацию и локализацию, которые являются средствами адаптации программного обеспечения к различным языкам, региональным различиям и техническим требованиям целевого рынка. Интернационализация — это процесс разработки программного приложения таким образом, чтобы его можно было адаптировать к различным языкам и регионам без значительных технических изменений. Локализация — это процесс адаптации интернационализированного программного обеспечения для определенного региона или языка путем

добавления компонентов, специфичных для локали, и перевода текста. Локализация и интернационализация должны учитывать такие факторы, как символы, числа, валюта, время и единицы измерения.

4.7 Метафоры и концептуальные модели

[17 , c5]

Разработчики пользовательского интерфейса могут использовать метафоры и концептуальные модели для установления сопоставлений между программным обеспечением и некоторой справочной системой, известной пользователям в реальном мире, что может помочь пользователям легче изучить и использовать интерфейс. Например, операцию «удалить файл» можно превратить в метафору с помощью значка мусорной корзины. При разработке пользовательского интерфейса инженеры-программисты должны быть осторожны, чтобы не использовать более одной метафоры для каждой концепции. Метафоры также представляют собой потенциальные проблемы в отношении интернационализации, поскольку не все метафоры имеют смысл или применяются одинаково во всех культурах.

5 Анализ и оценка качества разработки программного обеспечения

Этот раздел включает в себя ряд тем анализа качества и оценки, которые непосредственно связаны с проектированием программного обеспечения. (См. также KA качества программного обеспечения.)

5.1 Атрибуты качества

[4 , c4]

На качество разработки программного обеспечения влияют различные атрибуты, в том числе различные «-или» (сопровождаемость, переносимость, тестируемость, удобство использования) и «-есть» (правильность, надежность). Существует интересное различие между атрибутами качества, различимыми во время выполнения (например, производительность, безопасность, доступность, функциональность, удобство использования), теми, которые не различимы во время выполнения (например, модифицируемость, переносимость, возможность повторного использования, тестируемость), и атрибутами, связанными с архитектурой. внутренние качества (например, понятийная целостность, правильность, полнота). (См. также KA качества программного обеспечения.)

5.2 Методы анализа и оценки качества

[4 , c4] [5 , c24]

Различные инструменты и методы могут помочь в анализе и оценке качества разработки программного обеспечения.

- Обзоры дизайна программного обеспечения: неформальные и формализованные методы определения качества артефактов дизайна (например, обзоры архитектуры, обзоры дизайна и проверки; методы, основанные на сценариях; требования).

отслеживание). Обзоры дизайна программного обеспечения также могут оценивать безопасность. Можно просматривать вспомогательные материалы по установке, эксплуатации и использованию (например, руководства и файлы справки).

- Статический анализ: формальный или полуформальный статический (неисполняемый) анализ, который можно использовать для оценки проекта (например, анализ дерева отказов или автоматизированная перекрестная проверка). Анализ уязвимостей проекта (например,

статический анализ уязвимостей безопасности) может быть выполнен, если безопасность вызывает беспокойство. Формальный анализ проекта использует математические модели, которые позволяют разработчикам определять поведение и проверять производительность программного обеспечения вместо того, чтобы полностью полагаться на тестирование. Формальный анализ проекта можно использовать для обнаружения остаточных ошибок спецификации и дизайна (возможно, вызванных неточностью, двусмысленностью, а иногда и другими видами ошибок). (См. также Модели и методы разработки программного обеспечения KA.)

- Моделирование и прототипирование: динамические методы оценки проекта (например, моделирование производительности или прототипы осуществимости).

5.3 Меры

[4 , c4] [5 , c24]

Меры могут использоваться для оценки или количественной оценки различных аспектов дизайна программного обеспечения; например, размер, структура или качество. Большинство мер, которые были предложены, зависят от подхода, используемого для создания проекта. Эти меры подразделяются на две большие категории:

- Функциональные (структурированные) проектные меры: меры, полученные путем анализа функциональной декомпозиции; обычно представляется с помощью структурной диаграммы (иногда называемой иерархической диаграммой), на которой могут быть вычислены различные показатели.
- Показатели объектно-ориентированного проектирования: структура проекта обычно представляется в виде диаграммы классов, на которой могут быть вычислены различные показатели. Также могут быть вычислены показатели свойств внутреннего содержимого каждого класса.

6 обозначений дизайна программного обеспечения

Существует множество нотаций для представления артефактов проектирования программного обеспечения. Одни используются для описания структурной организации проекта, другие — для представления поведения программного обеспечения. Некоторые обозначения используются в основном при архитектурном проектировании, а другие - при детальном проектировании, хотя некоторые обозначения могут использоваться для обеих целей. Кроме того, некоторые обозначения используются в основном в контексте конкретных методов проектирования (см. тему 7, Стратегии и методы проектирования программного обеспечения). Обратите внимание, что проектирование программного обеспечения часто выполняется с использованием нескольких обозначений. Здесь они подразделяются на нотации для описания структурного (статического) представления и поведенческого (динамического) представления.

6.1 Структурные описания (статический вид)

[4 , c7] [5 , c6, c7] [6 , c4, c5, c6, c7] [12 , c7] [14 , c7]

Следующие обозначения, в основном, но не всегда графические, описывают и представляют структурные аспекты разработки программного обеспечения, то есть они используются для описания основных компонентов и того, как они взаимосвязаны (статическое представление):

- Языки описания архитектуры (ADL): текстовые, часто формальные языки, используемые для описания архитектуры программного обеспечения с точки зрения компонентов и соединителей.
- Диаграммы классов и объектов: используются для представления набора классов (и объектов) и их взаимосвязей.
- Диаграммы компонентов: используются для представления набора компонентов («физических и заменяемых частей системы, которые [соответствуют] и [обеспечивают] реализацию набора интерфейсов» [18]) и их взаимосвязей.
- Карты соавторов ответственности класса (CRC): используются для обозначения имен компонентов (класса), их обязанностей и имен их взаимодействующих компонентов.
- Диаграммы развертывания: используются для представления набора (физических) узлов и их взаимосвязей и, таким образом, для моделирования физических аспектов программного обеспечения.
- Диаграммы сущность-связь (ERD): используются для представления концептуальных моделей данных, хранящихся в информационных репозиториях.
- Языки описания интерфейса (IDL): языки, подобные программированию, используемые для определения интерфейсов (имен и типов экспортируемых операций) программных компонентов.
- Структурные диаграммы: используются для описания структуры вызовов программ (какие модули вызывают и вызываются, какие другие модули).

6.2 Поведенческие описания (динамический вид)

[4 , c7, c4s1, c13] [5 , c6, c7] [6 , c4, c5, c6, c7] [14 , c8]

Следующие обозначения и языки, некоторые графические, а некоторые текстовые, используются для описания динамического поведения программных систем и компонентов. Многие из этих обозначений полезны в основном, но не исключительно, при детальном проектировании. Более того, поведенческие описания могут включать в себя обоснование проектных решений, например, как проект будет соответствовать требованиям безопасности.

- Диаграммы действий: используются для отображения потока управления от действия к действию. Может использоваться для представления параллельных действий.
- Диаграммы связи: используются для отображения взаимодействий, происходящих между группой объектов; акцент делается на объектах, их ссылках и сообщениях, которыми они обмениваются по этим ссылкам.
- Диаграммы потоков данных (DFD): используются для отображения потока данных между элементами. Диаграмма потока данных обеспечивает «описание, основанное на моделировании потока информации вокруг сети операционных элементов, где каждый элемент использует или модифицирует информацию, поступающую в этот элемент» [4*]. Потоки данных (и, следовательно, диаграммы потоков данных) можно использовать для анализа безопасности, поскольку они предлагают идентификацию возможных путей атаки и раскрытия конфиденциальной информации.
- Таблицы решений и диаграммы: используются для представления сложных комбинаций условий и действий.
- Блок-схемы: используются для представления потока

управление и связанные с ним действия, которые необходимо выполнить.

- Диаграммы последовательности: используются для отображения взаимодействий между группой объектов с акцентом на временной порядок сообщений, передаваемых между объектами.
- Диаграммы перехода состояний и диаграммы состояний: используются для отображения потока управления из состояния в состояние и того, как поведение компонента изменяется в зависимости от его текущего состояния в конечном автомате.
- Языки формальных спецификаций: текстовые языки, которые используют основные понятия математики (например, логика, множество, последовательность) для строгого и абстрактного определения интерфейсов и поведения программных компонентов, часто в терминах пред- и постусловий. (См. также Модели и методы разработки программного обеспечения KA.)
- Языки псевдокода и разработки программ (PDL): структурированные языки, подобные языкам программирования, используемые для описания, как правило, на этапе детального проектирования, поведения процедуры или метода.

7 стратегий и методов проектирования программного обеспечения

Существуют различные общие стратегии, помогающие управлять процессом проектирования. В отличие от общих стратегий, методы более специфичны, поскольку они обычно предоставляют набор обозначений для использования с методом, описание процесса, который следует использовать при следовании методу, и набор рекомендаций по использованию метода. Такие методы полезны в качестве общей основы для групп разработчиков программного обеспечения. (См. также Модели и методы разработки программного обеспечения KA).

7.1 Общие стратегии

[4 , c8, c9, c10] [12 , c7]

Некоторые часто цитируемые примеры общих стратегий, полезных в процессе проектирования, включают стратегии «разделяй и властвуй» и стратегии пошагового уточнения, стратегии «сверху вниз» и «снизу вверх», а также стратегии, использующие эвристику, использование шаблонов и языков шаблонов, а также использование итеративного и поэтапного подхода.

7.2 Функционально-ориентированный (структурированный) дизайн

[4 , c13]

Это один из классических методов проектирования программного обеспечения, в котором декомпозиция сосредоточена на определении основных функций программного обеспечения, а затем на их разработке и уточнении в иерархическом порядке сверху вниз. Структурированный дизайн обычно используется

после структурного анализа, таким образом создавая (среди прочего) диаграммы потоков данных и связанные с ними описания процессов. Исследователи предложили различные стратегии (например, анализ трансформации, анализ транзакций) и эвристики (например, разветвление/разветвление, объем действия и объем контроля) для преобразования DFD в программную архитектуру, обычно представляемую в виде структурная схема.

7.3 Объектно-ориентированный дизайн

[4 , c16]

Было предложено множество методов проектирования программного обеспечения на основе объектов. Эта область эволюционировала от раннего объектно-ориентированного (ОО) дизайна середины 1980-х годов (существительное = объект; глагол = метод; прилагательное = атрибут), где ключевую роль играют наследование и полиморфизм, до области компонентного проектирования. , где метаинформация может быть определена и доступна (например, посредством отражения). Хотя корни объектно-ориентированного проектирования проистекают из концепции абстракции данных, проектирование, основанное на ответственности, было предложено в качестве альтернативного подхода к объектно-ориентированному проектированию.

7.4 Дизайн, ориентированный на структуру данных

[4 , c14, c15]

Проектирование, ориентированное на структуру данных, начинается со структур данных, которыми манипулирует программа, а не с функции, которую она выполняет. Инженер-программист сначала описывает структуры входных и выходных данных, а затем разрабатывает управляющую структуру программы на основе этих диаграмм структуры данных. Были предложены различные эвристики для обработки особых случаев, например, когда существует несоответствие между входной и выходной структурами.

7.5 Компонентный дизайн (CBD)

[4 , c17]

Программный компонент — это независимая единица с четко определенными интерфейсами и зависимостями, которые могут быть составлены и развернуты независимо. Компонентный дизайн решает проблемы, связанные с предоставлением, разработкой и интеграцией таких компонентов для улучшения повторного использования. Повторно используемые и готовые программные компоненты должны соответствовать тем же требованиям безопасности, что и новое программное обеспечение. Доверительное управление - это проблема дизайна; компоненты, созданные как имеющие определенную степень надежности, не должны зависеть от менее надежных компонентов или служб.

7.6 Другие методы

[5 , c19, c21]

Существуют и другие интересные подходы (см. Модели и методы разработки программного обеспечения KA). Итеративные и адаптивные методы реализуют инкременты программного обеспечения и уменьшают акцент на строгих требованиях к программному обеспечению и дизайну. Аспектно-ориентированный дизайн — это метод, с помощью которого программное обеспечение конструируется с использованием аспектов для реализации сквозных аспектов и расширений, которые определяются в процессе разработки требований к программному обеспечению. Сервис-ориентированная архитектура — это способ создания распределенного программного обеспечения с использованием веб-служб, выполняемых на распределенных компьютерах. Программные системы часто строятся с использованием служб от разных поставщиков, поскольку стандартные протоколы (такие как HTTP, HTTPS, SOAP) были разработаны для поддержки взаимодействия служб и обмена информацией о службах.

8 инструментов разработки программного обеспечения

[14 , c10, Приложение A]

Инструменты проектирования программного обеспечения можно использовать для поддержки создания артефактов проектирования программного обеспечения в процессе разработки программного обеспечения. Они могут частично или полностью поддерживать следующие виды деятельности:

- перевести модель требований в проектное представление;
- обеспечить поддержку представления функциональных компонентов и их интерфейса(ов);
- реализовать эвристическое уточнение и разбиение;
- дать рекомендации по оценке качества.

ДАЛЬНЕЙШИЕ ЧТЕНИЯ

Роджер Прессман, Программная инженерия: подход практика (седьмое издание) [19].

Примерно три десятилетия книга Роджера Прессмана «Разработка программного обеспечения: подход для практиков» была одним из ведущих мировых учебников по разработке программного обеспечения. Примечательно, что этот дополнительный учебник к [5*] всесторонне описывает дизайн программного обеспечения, включая концепции дизайна, архитектурный дизайн, дизайн на уровне компонентов, дизайн пользовательского интерфейса, дизайн на основе шаблонов и дизайн веб-приложений.

«Модель архитектуры 4 + 1 вид» [20].

Основополагающая статья «Модель представления 4+1» содержит описание архитектуры программного обеспечения с использованием пяти параллельных представлений. Четыре представления модели — это логическое представление, представление разработки, представление процесса и физическое представление. Кроме того, выбранные варианты использования или сценарии используются для иллюстрации архитектуры. Следовательно, модель содержит 4+1 представление. Представления используются для описания программного обеспечения в том виде, в каком его представляют различные заинтересованные стороны, такие как конечные пользователи, разработчики и менеджеры проектов.

Лен Басс, Пол Клементс и Рик Казман, Архитектура программного обеспечения на практике [21].

Эта книга знакомит с концепциями и передовым опытом архитектуры программного обеспечения, то есть структурирует программное обеспечение и взаимодействует его компоненты. Основываясь на собственном опыте, авторы охватывают основные технические темы проектирования, спецификации и проверки архитектур программного обеспечения. Они также подчеркивают важность бизнес-контекста, в котором разрабатывается большое программное обеспечение. Их цель — представить архитектуру программного обеспечения в реальных условиях, отражая как возможности, так и ограничения, с которыми сталкиваются организации. Это одна из лучших книг, доступных в настоящее время по архитектуре программного обеспечения.

ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

[1] ISO/IEC/IEEE., *24765:2010 Системная и программная инженерия. Словарь* , ISO/IEC/IEEE, 2010.

[2] IEEE Std., *12207-2008 (он же ISO/IEC 12207:2008) «Стандарт системной и программной инженерии — процессы жизненного цикла программного обеспечения»* , IEEE, 2008.

[3] Т. ДеМарко, «Парадокс архитектуры и дизайна программного обеспечения», *лекция премии Стивенса* , 1999 г.

[4] Д. Бадген, *Дизайн программного обеспечения* , 2-е изд., Addison-Wesley, 2003.

[5] И. Соммервиль, *Разработка программного обеспечения* , 9-е изд., Addison-Wesley, 2011.

[6] М. Пейдж-Джонс, *Основы объектно-ориентированного проектирования в UML* , 1-е изд., Addison-Wesley, 1999.

[7] Merriam-Webster, *Энциклопедический словарь Merriam-Webster* , 11-е изд., Merriam-Webster, 2003.

[8] IEEE, *стандарт IEEE. 1069-2009 Standard for Information Technology—Systems Design—Software Design Descriptions* , IEEE, 2009.

[9] ИСО/МЭК, *ИСО/МЭК 42010:2011 Системная и программная инженерия. Рекомендуемая практика для архитектурного описания программно-интенсивных систем* , ИСО/МЭК, 2011.

[10] Дж. Бош, *Проектирование и использование программных архитектур: внедрение и развитие линейного подхода к продуктам* , ACM Press, 2000.

[11] G. Kiczales и др., *Аспектно-ориентированное программирование, Proc. 11-я Европейская конф. Объектно-ориентированное программирование (ECOOP 97)* , Springer, 1997.

[12] Дж. Г. Брукшир, *Информатика: обзор* , 10-е изд., Addison-Wesley, 2008.

[13] Дж. Х. Аллен и др., *Разработка безопасности программного обеспечения: Руководство для менеджеров проектов* , Addison-Wesley, 2008.

[14] П. Клементс и др., *Документирование архитектуры программного обеспечения: взгляды и не только* , 2-е изд., Pearson Education, 2010.

[15] Э. Гамма и др., *Шаблоны проектирования: элементы многоразового объектно-ориентированного программного обеспечения* , 1-е изд., Addison-Wesley Professional, 1994.

[16] И. Джейкобсон, Г. Буч и Дж. Рамбо, *Унифицированный процесс разработки программного обеспечения* , Addison-Wesley Professional, 1999.

[17] Дж. Нильсен, *Юзабилити-инжиниринг* , Морган Кауфманн, 1993.

[18] Г. Буч, Дж. Рамбо и И. Джейкобсон, *Руководство пользователя унифицированного языка моделирования* , Addison-Wesley, 1999.

[19] Р. С. Прессман, *Программная инженерия: практический подход* , McGraw-Hill, 2010.

[20] П. Б. Крухтен, *Модель архитектуры 4+1 View* , IEEE Software, vol. 12, нет. 6, 1995, стр. 42-55.

[21] Л. Басс, П. Клементс и Р. Казман, *Архитектура программного обеспечения на практике* , 3-е изд., Addison-Wesley Professional, 2013.

- Последнее изменение этой страницы состоялось 24 августа 2015 г., в 19:51.