Глава 9: Модели программной инженерии

Из SWEBOK

Содержание

- 1 Моделирование
  - 1.1 Принципы моделирования
  - 1.2 Свойства и выражение моделей
  - 1.3 Синтаксис, семантика и прагматика
  - 1.4 Предусловия, постусловия и инварианты
- 2 типа моделей
  - 2.1 Информационное моделирование
  - 2.2 Поведенческое моделирование
  - 2.3 Моделирование конструкции
- 3 Анализ моделей
  - 3.1 Анализ полноты
  - 3.2 Анализ согласованности
  - 3.3 Анализ правильности
  - 3.4 Прослеживаемость
  - 3.5 Анализ взаимодействия
- 4 метода разработки программного обеспечения
  - 4.1 Эвристические методы
  - 4.2 Формальные методы
  - 4.3 Методы прототипирования
  - 4.4 Гибкие методы

АКРОНИМЫ

3GL	Язык третьего поколения
БНФ	Форма Бэкуса-Наура
СЗД	Разработка, ориентированная на функции
IDE	Интегрированная среда разработки
ПБИ	Элемент бэклога продукта
РАД	Быстрая разработка приложений
UML	Единый язык моделирования
XP	Экстремальное программирование

ВВЕДЕНИЕ

Модели и методы разработки программного обеспечения накладывают структуру на разработку программного обеспечения с целью сделать эту деятельность систематической, воспроизводимой и, в конечном итоге, более ориентированной на успех. Использование моделей обеспечивает подход к решению проблем, обозначения и процедуры построения и анализа моделей. Методы обеспечивают подход к систематической спецификации, проектированию, построению, тестированию и проверке

конечного программного обеспечения и связанных с ним рабочих продуктов. Модели и методы разработки программного обеспечения сильно различаются по своему охвату — от обращения к одной фазе жизненного цикла программного обеспечения до охвата всего жизненного цикла программного обеспечения. Акцент в этой области знаний (КА) делается на моделях и методах разработки программного обеспечения, которые охватывают несколько фаз жизненного цикла программного обеспечения, поскольку методы, специфичные для отдельных фаз жизненного цикла, охватываются другими КА.

РАЗБИВКА ТЕМ ПО МОДЕЛЯМ И МЕТОДАМ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Эта глава, посвященная моделям и методам разработки программного обеспечения, разделена на четыре основные тематические области:

- *Моделирование* : обсуждает общую практику моделирования и представляет темы принципов моделирования; свойства и выражение

модели; моделирование синтаксиса, семантики и прагматики; предусловия, постусловия и инварианты.

- *Типы моделей* : кратко обсуждаются модели и объединение подмоделей, а также приводятся некоторые общие характеристики типов моделей.

обычно встречается в практике разработки программного обеспечения.

- *Анализ моделей* : представлены некоторые из распространенных методов анализа, используемых при моделировании для проверки полноты, непротиворечивости, правильности, прослеживаемости и взаимодействия.
- *Методы разработки программного обеспечения* : представляет собой краткое изложение часто используемых методов разработки программного обеспечения. Обсуждение знакомит читателя с обзором эвристических методов, формальных методов, прототипирования и гибких методов.

Разбивка тем для моделей и методов программной инженерии КА показана на рис. 9.1.

1 Моделирование

Моделирование программного обеспечения становится широко распространенным методом, помогающим инженерам-программистам понимать, проектировать и сообщать аспекты программного обеспечения соответствующим заинтересованным сторонам. Заинтересованные стороны — это те лица или стороны, которые имеют заявленный или подразумеваемый интерес к программному обеспечению (например, пользователь, покупатель, поставщик, архитектор, орган по сертификации, оценщик, разработчик, инженер-программист и, возможно, другие). Хотя в литературе и на практике существует множество языков моделирования, нотаций, методов и инструментов, существуют унифицирующие общие концепции, которые в той или иной форме применимы ко всем из них. В следующих разделах приводятся общие сведения об этих общих концепциях.

1.1 Принципы моделирования

[1 , c2s2, c5s1, c5s2] [2 , c2s2][3 , c5s0]

Моделирование предоставляет инженеру-программисту организованный и систематический подход к представлению важных аспектов изучаемого программного обеспечения, облегчению принятия решений о программном обеспечении или его элементах и доведению этих важных решений до других заинтересованных сторон. Существуют три общих принципа, которыми руководствуются при моделировании:

- *Моделируйте основы* : хорошие модели обычно не отражают каждый аспект или функцию программного обеспечения при всех возможных условиях.

Моделирование обычно включает в себя разработку только тех аспектов или функций программного обеспечения, которые требуют конкретных ответов, абстрагируя любую второстепенную информацию. Такой подход делает модели управляемыми и полезными.

- *Обеспечить перспективу* : моделирование обеспечивает представления изучаемого программного обеспечения с использованием определенного набора правил для выражения

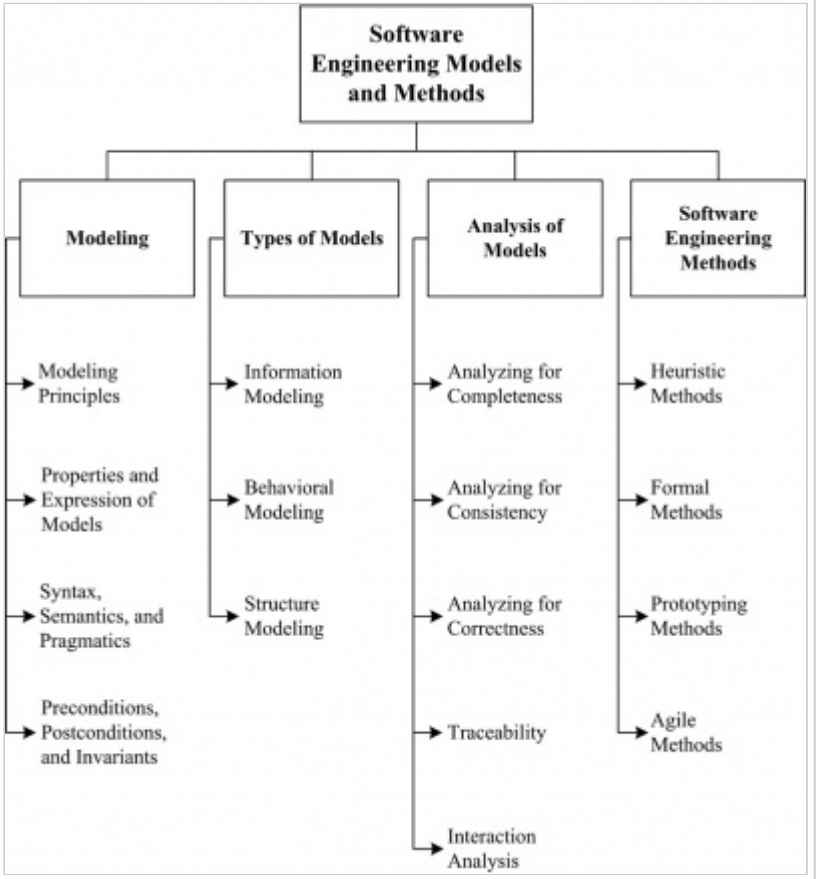

Рисунок 9.1: Разбивка тем для моделей и методов программной инженерии KA

модель в каждом представлении. Этот ориентированный на перспективу подход обеспечивает размерность модели (например, структурное представление, поведенческое представление, временное представление, организационное представление и другие представления по мере необходимости). Организация информации в виде фокусирует усилия по программному моделированию на конкретных задачах, относящихся к этому представлению, с использованием соответствующих обозначений, словаря, методов и инструментов.

- *Обеспечьте эффективную коммуникацию* : моделирование использует словарь предметной области программного обеспечения, язык моделирования и

семантическое выражение (другими словами, значение в контексте). При строгом и систематическом использовании это моделирование приводит к подходу к отчетности, который облегчает эффективную передачу информации о программном обеспечении заинтересованным сторонам проекта.

Модель — это *абстракция* или упрощение программного компонента. Следствием использования абстракции является то, что ни одна абстракция полностью не описывает программный компонент. Скорее, модель программного обеспечения представлена как совокупность абстракций, которые, взятые вместе, описывают только избранные аспекты, точки зрения или взгляды — только те, которые необходимы для принятия обоснованных решений и реагирования на причины создания модели в первое место. Это упрощение приводит к ряду предположений о контексте, в котором находится модель, которые также должны быть отражены в модели. Затем при повторном использовании модели эти допущения можно сначала проверить, чтобы установить актуальность повторно используемой модели в ее новом использовании и контексте.

1.2 Свойства и выражение моделей

[1 , c5s2, c5s3] [3 , c4s1.1p7, c4s6p3, c5s0p3]

Свойства моделей — это те отличительные черты конкретной модели, которые используются для характеристики ее полноты, непротиворечивости и правильности в рамках выбранной нотации моделирования и используемого инструментария. К свойствам моделей относятся следующие:

- *Полнота* : степень, в которой все требования были реализованы и проверены в рамках модели.
- *Согласованность* : степень, в которой модель не содержит противоречивых требований, утверждений, ограничений, функций или компонентов.

описания.

- *Правильность* : степень, в которой модель удовлетворяет требованиям и спецификациям проекта и не имеет дефектов.

Модели создаются для представления объектов реального мира и их поведения, чтобы ответить на конкретные вопросы о том, как программное обеспечение должно работать. Опрос моделей — путем исследования, моделирования или обзора — может выявить области неопределенности в модели и программном обеспечении, на которое ссылается модель. Эти неопределенности или оставшиеся без ответа вопросы, касающиеся требований, дизайна и/или реализации, могут быть соответствующим образом обработаны. Основным элементом выражения модели является сущность. Сущность может представлять конкретные артефакты (например, процессоры, датчики или роботы) или абстрактные артефакты (например, программные модули или протоколы связи). Объекты модели связаны с другими объектами с помощью отношений (другими словами, строк или текстовых операторов над целевыми объектами). Выражение объектов модели может быть выполнено с использованием текстовых или графических языков моделирования; оба типа языка моделирования соединяют объекты модели через определенные языковые конструкции. Значение объекта может быть представлено его формой, текстовыми атрибутами или тем и другим. Как правило, текстовая информация придерживается синтаксической структуры, характерной для языка. Точные значения, связанные с моделированием контекста, структуры или поведения с использованием этих сущностей и отношений, зависят от используемого языка моделирования, строгости проектирования, применяемой к усилиям по моделированию, конкретного конструируемого представления и сущности, к которой относится конкретная нотация. элемент можно прикрепить. Для захвата необходимой семантики программного обеспечения может потребоваться несколько представлений модели. При использовании моделей, поддерживаемых автоматизацией, модели могут быть проверены на полноту и непротиворечивость. Полезность этих проверок во многом зависит от уровня семантической и синтаксической строгости, применяемой к усилиям по моделированию в дополнение к явной инструментальной поддержке. Правильность обычно проверяется путем моделирования и/или проверки.

1.3 Синтаксис, семантика и прагматика

[2 , c2s2.2.2p6] [3 , c5s0]

Модели могут быть удивительно обманчивы. Тот факт, что модель является абстракцией с недостающей информацией, может привести к ложному пониманию того, что программное обеспечение можно полностью понять по одной модели. Полная модель («полная» по отношению к усилиям по моделированию) может быть объединением нескольких подмоделей и любых моделей специальных функций. Изучение и принятие решений относительно одной модели в этом наборе подмоделей могут быть проблематичными. Понимание точного значения конструкций моделирования также может быть затруднено. Языки моделирования определяются синтаксическими и семантическими правилами. Для текстовых языков синтаксис определяется с помощью грамматики обозначений, которая определяет допустимые языковые конструкции (например, форма Бэкуса-Наура (BNF)). Для графических языков синтаксис определяется с помощью графических моделей, называемых метамоделями. Как и в случае с БНФ, метамодели определяют действительные синтаксические конструкции языка графического моделирования; метамодель определяет, как эти конструкции могут быть составлены для создания действительных моделей. Семантика для языков моделирования определяет значение, придаваемое сущностям и отношениям, зафиксированным в модели. Например, простая диаграмма двух ящиков, соединенных линией, может быть интерпретирована по-разному. Знание того, что диаграмма, на которой расположены и соединены блоки, является диаграммой объектов или диаграммой деятельности, может помочь в интерпретации этой модели. На практике обычно имеется хорошее понимание семантики конкретной программной модели благодаря выбранному языку моделирования, тому, как этот язык моделирования используется для выражения сущностей и отношений в этой модели, опыту разработчиков моделей. , и контекст, в котором моделирование было предпринято и таким образом представлено. Смысл передается через модель даже при наличии неполной информации посредством абстракции; прагматика объясняет, как смысл воплощается в модели и ее контексте и эффективно передается другим разработчикам программного обеспечения. Однако все еще есть случаи, когда необходима осторожность в отношении моделирования и семантики. Например, любые части модели, импортированные из другой модели или библиотеки, должны быть проверены на наличие семантических допущений, конфликтующих в новой среде моделирования; это может быть неочевидно. Модель должна быть проверена на наличие задокументированных допущений. Хотя синтаксис моделирования может быть идентичным, модель может означать что-то совсем другое в новой среде, то есть в другом

контексте. Кроме того, учтите, что по мере развития программного обеспечения и внесения изменений могут быть внесены семантические разногласия, приводящие к ошибкам. Поскольку многие инженеры-программисты работают над частью модели с течением времени в сочетании с обновлениями инструментов и, возможно, новыми требованиями, есть возможности для частей модели представлять что-то отличное от первоначального намерения автора и исходного контекста модели.

1.4 Предусловия, постусловия и инварианты

[2 , c4s4] [4 , c10s4p2, c10s5p2p4]

При моделировании функций или методов инженер-программист обычно начинает с набора предположений о состоянии программного обеспечения до, во время и после выполнения функции или метода. Эти допущения необходимы для правильной работы функции или метода и сгруппированы для обсуждения в виде набора предусловий, постусловий и инвариантов.

- *Предварительные условия* : набор условий, которые должны быть выполнены до выполнения функции или метода. Если эти предварительные условия не выполняются до выполнения функции или метода, функция или метод могут давать ошибочные результаты.
- *Постусловия* : набор условий, выполнение которых гарантировано после успешного выполнения функции или метода. Как правило, постусловия представляют, как изменилось состояние программного обеспечения, как изменились параметры, переданные функции или методу, как изменились значения данных или как было затронуто возвращаемое значение.
- *Инварианты* : набор условий в операционной среде, которые сохраняются (другими словами, не меняются) до и после

выполнение функции или метода. Эти инварианты важны и необходимы для программного обеспечения и правильной работы функции или метода.

2 типа моделей

Типичная модель состоит из совокупности подмоделей. Каждая подмодель является частичным описанием и создается для определенной цели; он может состоять из одной или нескольких диаграмм. Набор подмоделей может использовать несколько языков моделирования или один язык моделирования. Унифицированный язык моделирования (UML) распознает богатую коллекцию диаграмм моделирования. Использование этих диаграмм вместе с конструкциями языка моделирования приводит к трем широко используемым типам моделей: информационным моделям, поведенческим моделям и структурным моделям (см. раздел 1.1).

2.1 Информационное моделирование

[1 , c7s2.2] [3 , c8s1]

Информационные модели обеспечивают центральное внимание к данным и информации. Информационная модель — это абстрактное представление, которое идентифицирует и определяет набор концепций, свойств, отношений и ограничений для объектов данных. Семантическая или концептуальная информационная модель часто используется для придания определенного формализма и контекста моделируемому программному обеспечению с точки зрения проблемы, не заботясь о том, как эта модель фактически отображается на реализацию программного обеспечения. Семантическая или концептуальная информационная модель является абстракцией и, как таковая, включает только концепции, свойства, отношения и ограничения, необходимые для концептуализации реального представления информации.

2.2 Поведенческое моделирование

[1 , c7s2.1, c7s2.3, c7s2.4] [2 , c9s2] [3 , c5s4]

Поведенческие модели идентифицируют и определяют функции моделируемого программного обеспечения. Поведенческие модели обычно принимают три основные формы: конечные автоматы, модели потока управления и модели потока данных. Конечные автоматы обеспечивают модель программного обеспечения как набор определенных состояний, событий и переходов. Программное обеспечение переходит из одного состояния в другое посредством защищенного или незащищенного запускающего события, которое происходит в моделируемой среде. Модели потока управления

показывают, как последовательность событий вызывает активацию или деактивацию процессов. Поведение потока данных описывается как последовательность шагов, в ходе которых данные проходят через процессы к хранилищам или приемникам данных.

2.3 Моделирование конструкции

[1 , c7s2.5, c7s3.1, c7s3.2] [3 , c5s3] [4 , c4]

Структурные модели иллюстрируют физическую или логическую композицию программного обеспечения из его различных составных частей. Моделирование структуры устанавливает определенную границу между реализуемым или моделируемым программным обеспечением и средой, в которой оно должно работать. Некоторыми общими структурными конструкциями, используемыми при структурном моделировании, являются композиция, декомпозиция, обобщение и специализация сущностей; выявление релевантных отношений и кардинальности между сущностями; и определение процессов или функциональных интерфейсов. Структурные диаграммы, предоставляемые UML для структурного моделирования, включают диаграммы классов, компонентов, объектов, развертывания и упаковки.

3 Анализ моделей

Разработка моделей дает инженеру-программисту возможность изучать, рассуждать и понимать структуру, функции, операционное использование и аспекты сборки, связанные с программным обеспечением. Анализ построенных моделей необходим, чтобы гарантировать, что эти модели являются полными, непротиворечивыми и достаточно правильными, чтобы служить своей цели для заинтересованных сторон. В следующих разделах кратко описываются методы анализа, обычно используемые с моделями программного обеспечения, чтобы гарантировать, что инженер-программист и другие заинтересованные стороны получат соответствующую выгоду от разработки и использования моделей.

3.1 Анализ полноты

[3 , c4s1.1p7, c4s6] [5 , p8–11]

Чтобы иметь программное обеспечение, полностью отвечающее потребностям заинтересованных сторон, крайне важна полнота — от процесса выявления требований до реализации кода. Полнота – это степень, в которой все указанные требования были реализованы и проверены. Модели могут быть проверены на полноту с помощью инструмента моделирования, который использует такие методы, как структурный анализ и анализ достижимости в пространстве состояний (которые гарантируют, что все пути в моделях состояний достигаются с помощью некоторого набора правильных входных данных); модели также могут быть проверены на полноту вручную с помощью инспекций или других методов проверки (см. КА качества программного обеспечения). Ошибки и предупреждения, генерируемые этими инструментами анализа и обнаруживаемые при проверке или обзоре, указывают на вероятные необходимые корректирующие действия для обеспечения полноты моделей.

3.2 Анализ согласованности

[3 , c4s1.1p7, c4s6] [5 , p8–11]

Согласованность — это степень, в которой модели не содержат противоречивых требований, утверждений, ограничений, функций или описаний компонентов. Обычно проверка непротиворечивости выполняется с помощью инструмента моделирования с использованием функции автоматического анализа; модели также могут быть проверены на согласованность вручную с использованием проверок или других методов проверки (см. КА качества программного обеспечения). Как и в случае с полнотой, ошибки и предупреждения, генерируемые этими инструментами анализа и обнаруживаемые при проверке или обзоре, указывают на необходимость корректирующих действий.

3.3 Анализ правильности

[5 , стр. 8–11]

Корректность — это степень, в которой модель удовлетворяет требованиям к программному обеспечению и спецификациям проектирования программного обеспечения, не содержит дефектов и в конечном итоге удовлетворяет потребности заинтересованных сторон. Анализ правильности включает в себя проверку синтаксической правильности модели (то есть правильное использование грамматики и

конструкций языка моделирования) и проверку семантической правильности модели (то есть использование конструкций языка моделирования для правильного представления значения того, что является моделируется). Чтобы проанализировать модель на синтаксическую и семантическую правильность, ее анализируют — либо автоматически (например, с помощью инструмента моделирования для проверки синтаксической правильности модели), либо вручную (используя проверки или другие методы проверки) — отыскивая возможные дефекты, а затем удаляя или устранение подтвержденных дефектов до того, как программное обеспечение будет выпущено для использования.

3.4 Прослеживаемость

[3 , c4s7.1, c4s7.2]

Разработка программного обеспечения обычно включает использование, создание и модификацию многих рабочих продуктов, таких как документы планирования, спецификации процессов, требования к программному обеспечению, диаграммы, проекты и псевдокод, написанный от руки и сгенерированный инструментами код, ручные и автоматические тестовые сценарии и отчеты, а также файлы и данные. Эти рабочие продукты могут быть связаны через различные отношения зависимости (например, использование, реализация и тесты). Поскольку программное обеспечение разрабатывается, управляется, поддерживается или расширяется, необходимо отображать и контролировать эти отношения трассируемости, чтобы продемонстрировать согласованность требований к программному обеспечению с моделью программного обеспечения (см. Отслеживание требований в требованиях к программному обеспечению KA) и многими рабочими продуктами. Использование прослеживаемости обычно улучшает управление программными продуктами и качеством программных процессов; он также предоставляет заинтересованным сторонам гарантии того, что все требования были выполнены. Прослеживаемость позволяет проводить анализ изменений после разработки и выпуска программного обеспечения, поскольку можно легко отслеживать взаимосвязи с рабочими программными продуктами для оценки влияния изменений. Инструменты моделирования обычно предоставляют некоторые автоматизированные или ручные средства для определения и управления связями прослеживаемости между требованиями, дизайном, кодом и/или объектами тестирования, которые могут быть представлены в моделях и других программных рабочих продуктах. (Для получения дополнительной информации об отслеживаемости см. KA управления конфигурацией программного обеспечения). Инструменты моделирования обычно предоставляют некоторые автоматизированные или ручные средства для определения и управления связями прослеживаемости между требованиями, дизайном, кодом и/или объектами тестирования, которые могут быть представлены в моделях и других программных рабочих продуктах. (Для получения дополнительной информации об отслеживаемости см. KA управления конфигурацией программного обеспечения). Инструменты моделирования обычно предоставляют некоторые автоматизированные или ручные средства для определения и управления связями прослеживаемости между требованиями, дизайном, кодом и/или объектами тестирования, которые могут быть представлены в моделях и других программных рабочих продуктах. (Для получения дополнительной информации об отслеживаемости см. KA управления конфигурацией программного обеспечения).

3.5 Анализ взаимодействия

[2 , c10, c11] [3 , c29s1.1, c29s5] [4 , c5]

Анализ взаимодействия фокусируется на коммуникациях или отношениях потока управления между объектами, используемыми для выполнения конкретной задачи или функции в модели программного обеспечения. Этот анализ исследует динамическое поведение взаимодействий между различными частями модели программного обеспечения, включая другие уровни программного обеспечения (такие как операционная система, промежуточное программное обеспечение и приложения). Для некоторых программных приложений также может быть важно изучить взаимодействие между компьютерным программным приложением и программным обеспечением пользовательского интерфейса. Некоторые среды моделирования программного обеспечения предоставляют средства моделирования для изучения аспектов динамического поведения моделируемого программного обеспечения.

4 метода разработки программного обеспечения

Методы программной инженерии обеспечивают организованный и систематический подход к разработке программного обеспечения для целевого компьютера. Существует множество методов, из которых можно выбирать, и для инженера-программиста важно выбрать подходящий метод или методы для

стоящей перед ним задачи разработки программного обеспечения; этот выбор может оказать существенное влияние на успех программного проекта. Использование этих методов разработки программного обеспечения в сочетании с людьми с нужным набором навыков и инструментами позволяет разработчикам программного обеспечения визуализировать детали программного обеспечения и в конечном итоге преобразовать представление в рабочий набор кода и данных. Отдельные методы разработки программного обеспечения обсуждаются ниже. Тематические области организованы в обсуждение эвристических методов, формальных методов, методов прототипирования и гибких методов.

4.1 Эвристические методы

[1 , c13, c15, c16] [3 , c2c2.2, c5c4.1, c7c1]

Эвристические методы — это методы разработки программного обеспечения, основанные на опыте, которые были и довольно широко применяются в индустрии программного обеспечения. Эта тематическая область содержит три широкие категории для обсуждения: методы структурного анализа и проектирования, методы моделирования данных и методы объектно-ориентированного анализа и проектирования.

- *Методы структурированного анализа и проектирования* : модель программного обеспечения разрабатывается в первую очередь с функциональной или поведенческой точки зрения, начиная с высокоуровневого представления программного обеспечения (включая данные и элементы управления), а затем постепенно разлагая или уточняя компоненты модели посредством все более детального проектирования. Детальный проект в конечном итоге сводится к очень конкретным деталям или спецификациям программного обеспечения, которое должно быть закодировано (вручную, автоматически или и то, и другое), построено, протестировано и проверено.
- *Методы моделирования данных* : модель данных строится с точки зрения используемых данных или информации. Таблицы данных и отношения

определить модели данных. Этот метод моделирования данных используется в основном для определения и анализа требований к данным, поддерживающих проекты баз данных или репозиториев данных, которые обычно используются в бизнес-программном обеспечении, где данные активно управляются как ресурс или актив бизнес-системы.

- *Объектно-ориентированный анализ и методы проектирования* . Объектно-ориентированная модель представлена как набор объектов, которые инкапсулируют

данные и отношения и взаимодействовать с другими объектами с помощью методов. Объекты могут быть предметами реального мира или виртуальными предметами. Модель программного обеспечения строится с использованием диаграмм, составляющих выбранные представления программного обеспечения. Постепенное совершенствование моделей программного обеспечения приводит к детальному проектированию. Затем детальный проект либо развивается посредством последовательных итераций, либо преобразуется (с использованием некоторого механизма) в представление реализации модели, где выражается код и подход к упаковке для окончательного выпуска и развертывания программного продукта.

4.2 Формальные методы

[1 , c18] [3 , c27] [5 , c8-c24]

Формальные методы — это методы разработки программного обеспечения, используемые для спецификации, разработки и проверки программного обеспечения посредством применения строгих математических обозначений и языка. Благодаря использованию языка спецификаций модель программного обеспечения может быть проверена на непротиворечивость (другими словами, на отсутствие двусмысленности), полноту и правильность систематическим и автоматизированным или полуавтоматическим способом. Эта тема связана с разделом «Формальный анализ» в KA «Требования к программному обеспечению». В этом разделе рассматриваются языки спецификаций, уточнение и вывод программ, формальная проверка и логический вывод.

- *Языки спецификаций* : Языки спецификаций обеспечивают математическую основу для формального метода; языки спецификаций — это формальные компьютерные языки более высокого уровня (другими словами, не классический язык программирования 3-го поколения

(3GL)), используемые на этапах спецификации программного обеспечения, анализа требований и/или проектирования для описания конкретного поведения ввода/вывода. Языки спецификаций не являются непосредственно исполняемыми языками; обычно они состоят из нотации и синтаксиса, семантики использования нотации и набора допустимых отношений для объектов.
- *Уточнение и вывод* программы. Уточнение программы — это процесс создания спецификации более низкого уровня (или более подробной) с использованием ряда преобразований. Именно посредством последовательных преобразований инженер-программист получает исполняемое представление.

программы. Спецификации можно уточнять, добавляя детали до тех пор, пока модель не будет сформулирована на языке программирования 3GL или в исполняемой части выбранного языка спецификации. Это уточнение спецификации стало возможным благодаря определению спецификаций с точными семантическими свойствами; спецификации должны устанавливать не только отношения между сущностями, но и точное значение этих отношений и операций во время выполнения.

- *Формальная проверка* : проверка модели — это формальный метод проверки; это обычно включает в себя выполнение исследования пространства состояний

или анализ достижимости, чтобы продемонстрировать, что представленный дизайн программного обеспечения имеет или сохраняет определенные интересующие свойства модели. Примером проверки модели является анализ, который проверяет правильность поведения программы при всех возможных чередованиях событий или поступлений сообщений. Использование формальной верификации требует строго определенной модели программного обеспечения и его операционной среды; эта модель часто принимает форму конечного автомата или другого формально определенного автомата.

- *Логический вывод* : логический вывод — это метод разработки программного обеспечения, который включает указание предварительных и постусловий.

вокруг каждого значимого блока проекта и — используя математическую логику — разработать доказательство того, что эти предварительные и постусловия должны выполняться при всех входных данных. Это дает инженеру-программисту возможность предсказать поведение программного обеспечения, не запуская его. Некоторые интегрированные среды разработки (IDE) включают способы представления этих доказательств вместе с дизайном или кодом.

4.3 Методы прототипирования

[1 , c12s2] [3 , c2s3.1] [6 , c7s3p5]

Прототипирование программного обеспечения — это деятельность, которая обычно создает неполные или минимально функциональные версии программного приложения, обычно для опробования конкретных новых функций, получения отзывов о требованиях к программному обеспечению или пользовательскому интерфейсу, дальнейшего изучения требований к программному обеспечению, дизайна программного обеспечения или вариантов реализации и/или получить некоторые другие полезные сведения о программном обеспечении. Инженер-программист выбирает метод прототипирования, чтобы сначала понять наименее понятные аспекты или компоненты программного обеспечения; этот подход отличается от других методов разработки программного обеспечения, которые обычно начинают разработку с наиболее понятных частей. Как правило, прототип продукта не становится окончательным программным продуктом без обширной доработки или рефакторинга. В этом разделе обсуждаются стили прототипирования, цели,

- *Стиль прототипирования* : здесь рассматриваются различные подходы к разработке прототипов. Прототипы могут разрабатываться как одноразовый код или бумажные продукты, как эволюция рабочего проекта или как исполняемая спецификация. Для каждого стиля обычно используются разные процессы жизненного цикла прототипирования. Выбранный стиль зависит от типа результатов, необходимых проекту, качества необходимых результатов и срочности результатов.
- *Цель* создания прототипа: целью деятельности по созданию прототипа является конкретный продукт, который обслуживается усилиями по созданию прототипа. Примеры целей прототипирования включают спецификацию требований, элемент или компонент архитектурного дизайна, алгоритм или человеко-машинный пользовательский интерфейс.
- *Методы оценки прототипирования* : инженер-программист или другие участники проекта могут использовать или оценивать прототип различными способами, в первую очередь руководствуясь

глубинными причинами, которые привели к разработке прототипа. Прототипы могут быть оценены или протестированы в сравнении с фактически реализованным программным обеспечением или в отношении целевого набора требований (например, прототип требований); прототип может также служить моделью для будущей разработки программного обеспечения (например, как в спецификации пользовательского интерфейса).

4.4 Гибкие методы

[3 , c3] [6 , c7c3п7] [7 , c6, Приб.A]

Agile-методы родились в 1990-х годах из-за необходимости уменьшить очевидные большие накладные расходы, связанные с тяжеловесными плановыми методами, используемыми в крупномасштабных проектах разработки программного обеспечения. Agile-методы считаются облегченными методами, поскольку они характеризуются короткими итеративными циклами разработки, самоорганизующимися командами, более простым дизайном, рефакторингом кода, разработкой через тестирование, частым участием клиентов и акцентом на создание наглядного рабочего продукта при каждой разработке. цикл. В литературе доступно множество гибких методов; некоторые из наиболее популярных подходов, которые кратко обсуждаются здесь, включают быструю разработку приложений (RAD), экстремальное программирование (XP), Scrum и разработку, основанную на функциях (FDD).

- *РАД* : Методы быстрой разработки программного обеспечения используются в основном при разработке приложений для бизнес-систем, требующих больших объемов данных. Метод RAD реализуется с помощью специальных инструментов разработки баз данных, используемых разработчиками программного обеспечения для быстрой разработки, тестирования и развертывания новых или модифицированных бизнес-приложений.
- *XP* : этот подход использует истории или сценарии для требований, сначала разрабатывает тесты, имеет прямое участие клиента в команде (обычно определяет приемочные тесты), использует парное программирование и обеспечивает непрерывный рефакторинг кода и интеграцию. Истории разбиваются на задачи, расставляются по приоритетам, оцениваются, разрабатываются и тестируются. Каждое обновление программного обеспечения тестируется с помощью автоматических и ручных тестов; приращение может выпускаться часто, например, каждые пару недель или около того.
- *Scrum* : этот гибкий подход более удобен для управления проектами, чем другие. Скрам-мастер управляет действиями в рамках приращения проекта; каждый инкремент называется спринтом и длится не более 30 дней. Разрабатывается список элементов невыполненной работы по продукту (PBI), на основе которого идентифицируются, определяются, расставляются приоритеты и оцениваются задачи. Рабочая версия программного обеспечения тестируется и выпускается в каждом инкременте. Ежедневные скрам-встречи обеспечивают планирование работы.
- *FDD* : это основанный на модели, короткий, итеративный подход к разработке программного обеспечения, использующий пятиэтапный процесс: (1) разработать модель продукта, чтобы охватить широту предметной области, (2) создать список потребностей или функций, (3) построить план разработки функций, (4) разработать дизайн для функций, специфичных для итерации, и (5) закодировать, протестировать, а затем интегрировать функции. FDD похож на инкрементный подход к разработке программного обеспечения; он также похож на XP, за исключением того, что право собственности на код назначается отдельным лицам, а не команде. FDD подчеркивает общий архитектурный подход к программному обеспечению, который способствует правильному созданию функции с первого раза, а не постоянному рефакторингу.

There are many more variations of agile methods in the literature and in practice. Note that there will always be a place for heavyweight, plan-based software engineering methods as well as places where agile methods shine. There are new methods arising from combinations of agile and plan-based methods where practitioners are defining new methods that balance the features needed in both heavyweight and lightweight methods based primarily on prevailing organizational business needs. These business needs, as typically represented by some of the project stakeholders, should and do drive the choice in using one software engineering method over another or in constructing a new method from the best features of a combination of software engineering methods.

REFERENCES

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

[2] SJ Mellor и MJ Balcer, *Executable UML: A Foundation for Model-Driven Architecture* , 1-е изд., Addison-Wesley, 2002.

[3] И. Соммервиль, *Программная инженерия* , 9-е изд., Эддисон Уэсли, 2011.

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

[5] Дж. М. Винг, «Введение спецификатора в формальные методы», Computer, vol. 23, нет. 9, 1990, стр. 8, 10–23.

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

[7] Б. Бем и Р. Тернер, *Баланс ловкости и дисциплины: Руководство для растерянных* , Addison-Wesley, 2003.