Содержание

Введение 11

Цели 11
Для кого предназначена эта книга 12
Как работать с этой книгой 12
Организация книги и особенности изложения материала 13
Краткая история UML 14

Часть I Введение в процесс моделирования 17

Глава 1. Зачем мы моделируем 18

Значение моделирования 19
Принципы моделирования 23
Объектное моделирование 26

Глава 2. Введение в UML 28

Обзор UML 28
Концептуальная модель UML 32
Архитектура 47
Жизненный цикл разработки программного обеспечения 50

Глава 3. Здравствуй, мир! 53

Ключевые абстракции 53
Механизмы 57
Артефакты 58

Часть II Основы структурного моделирования 61

Глава 4. Классы 62

Введение 62
Базовые понятия 64
Типичные приемы моделирования 69
Советы и подсказки 74

Глава 5. Связи 75

Введение 76
Базовые понятия 77

Типичные приемы моделирования 83
Советы и подсказки 88

Глава 6. Общие механизмы 90

Введение 91
Базовые понятия 93
Типичные приемы моделирования 100
Советы и подсказки 103

Глава 7. Диаграммы 105

Базовые понятия 107
Типичные приемы моделирования 112
Советы и подсказки 118

Глава 8. Диаграммы классов 120

Введение 120
Базовые понятия 122
Типичные приемы моделирования 123
Советы и подсказки 130

Часть III Расширенное структурное моделирование 133

Глава 9. Расширенные классы 134

Введение 134
Базовые понятия 135
Типичные приемы моделирования 147
Советы и подсказки 148

Глава 10. Расширенные связи 150

Введение 150
Базовые понятия 152
Типичные приемы моделирования 165
Советы и подсказки 166

Глава 11. Интерфейсы, типы и роли 167

Введение 167
Базовые понятия 169
Типичные приемы моделирования 173
Советы и подсказки 177

Глава 12. Пакеты 178

Введение 178
Базовые понятия 179
Типичные приемы моделирования 185
Советы и подсказки 188

Глава 13. Экземпляры ... 190

Введение ... 190
Базовые понятия ... 191
Типичные приемы моделирования ... 197
Советы и подсказки ... 198

Глава 14. Диаграммы объектов ... 199

Введение ... 199
Базовые понятия ... 201
Типичные приемы моделирования ... 202
Советы и подсказки ... 205

Глава 15. Компоненты ... 206

Введение ... 206
Базовые понятия ... 207
Типичные приемы моделирования ... 217
Советы и подсказки ... 219

Часть IV Основы моделирования поведения ... 221

Глава 16. Взаимодействия ... 222

Введение ... 222
Базовые понятия ... 224
Типичные приемы моделирования ... 234
Советы и подсказки ... 236

Глава 17. Варианты использования ... 238

Введение ... 238
Базовые понятия ... 241
Типичные приемы моделирования ... 249
Советы и подсказки ... 251

Глава 18. Диаграммы вариантов использования ... 252

Введение ... 252
Базовые понятия ... 254
Типичные приемы моделирования ... 255
Советы и подсказки ... 261

Глава 19. Диаграммы взаимодействия ... 262

Введение ... 263
Базовые понятия ... 264
Типичные приемы моделирования ... 274

Глава 20. Диаграммы деятельности ... 281

Введение ... 282
Базовые понятия ... 283
Типичные приемы моделирования ... 294
Советы и подсказки ... 299

Часть V Расширенное моделирование поведения ... 301

Глава 21. События и сигналы ... 302

Введение ... 302
Базовые понятия ... 303
Типичные приемы моделирования ... 308
Советы и подсказки ... 311

Глава 22. Конечные автоматы ... 312

Введение ... 313
Термины и понятия ... 314
Типичные приемы моделирования ... 332
Советы и подсказки ... 335

Глава 23. Процессы и потоки ... 337

Введение ... 338
Базовые понятия ... 339
Типичные приемы моделирования ... 345
Советы и подсказки ... 348

Глава 24. Время и пространство ... 349

Введение ... 349
Базовые понятия ... 350
Типичные приемы моделирования ... 353
Советы и подсказки ... 356

Глава 25. Диаграммы состояний ... 357

Введение ... 358
Базовые понятия ... 359
Типичные приемы моделирования ... 361
Советы и подсказки ... 366

Часть VI Моделирование архитектуры ... 367

Глава 26. Артефакты ... 368

Введение ... 368
Базовые понятия ... 369

Типичные приемы моделирования ... 372
Советы и подсказки ... 377

Глава 27. Размещение ... 379

Введение ... 379
Базовые понятия ... 380
Типичные приемы моделирования ... 384
Советы и подсказки ... 386

Глава 28. Кооперации ... 387

Введение ... 387
Базовые понятия ... 389
Типичные приемы моделирования ... 394
Советы и подсказки ... 400

Глава 29. Образцы и каркасы ... 401

Введение ... 401
Базовые понятия ... 403
Типичные приемы моделирования ... 407
Советы и подсказки ... 412

Глава 30. Диаграммы артефактов ... 413

Введение ... 413
Термины и понятия ... 414
Типичные приемы моделирования ... 416
Советы и подсказки ... 426

Глава 31. Диаграммы размещения ... 427

Введение ... 427
Базовые понятия ... 429
Типичные приемы моделирования ... 431
Советы и подсказки ... 437

Глава 32. Системы и модели ... 439

Введение ... 439
Термины и понятия ... 441
Типичные приемы моделирования ... 444

Часть VII Итоги ... 449

Глава 33. Применение UML ... 450

Переход к UML ... 450
Что дальше ... 452

Приложение 1. Нотация UML 454

Сущности 454
Связи 457
Расширяемость 458
Диаграммы 458

Приложение 2. Rational Unified Process 460

Характеристики процесса 460
Фазы и итерации 462
Дисциплины 465
Рабочие продукты 466

Глоссарий 469

Предметный указатель 483

Введение

Унифицированный язык моделирования (Unified Modeling Language, UML) – это графический язык для визуализации, специфицирования, конструирования и документирования систем, в которых главная роль принадлежит программному обеспечению. С помощью UML можно разработать детальный план создаваемой системы, содержащий не только ее концептуальные элементы, такие как системные функции и бизнес-процессы, но и конкретные особенности, например классы, написанные на каком-либо языке программирования, схемы баз данных и повторно используемые программные компоненты.

Эта книга научит вас эффективно использовать UML. В ней рассматривается версия UML 2.0.

Цели

Прочитав эту книгу, вы:

- научитесь различать, в чем может, а в чем не может пригодиться UML, а также поймете, почему этот язык важен в процессе разработки сложных программных систем;
- освоите словарь, правила, идиомы языка UML и, самое главное, научитесь «бегло говорить» на нем;
- поймете, как можно использовать UML для решения разнообразных проблем моделирования.

Предлагаемое вашему вниманию руководство пользователя описывает специфические свойства языка UML. Тем не менее книга не задумывалась как исчерпывающее справочное руководство по UML. Эту функцию выполняет справочник Rumbaugh, Jacobson, Booch. The Unified Modeling Language Reference Manual. Second Edition (Addison-Wesley, 2005)1.

Предлагаемое вашему вниманию руководство описывает процесс разработки программных систем с использованием UML, однако не предоставляет полную информацию об этом процессе. Более подробные сведения приводятся в книге: Rumbaugh, Jacobson, Booch. Unified Software Development Process. Addison-Wesley,1999.

[1] Русскоязычное издание книги: Буч Г., Якобсон А., Рамбо Дж. UML. 2-е изд. – СПб.: Питер, 2006.

Наконец, в книге содержится множество рекомендаций и советов по использованию UML для решения часто возникающих задач моделирования, но моделированию как таковому она не учит. Подобный подход принят в руководствах по большинству языков программирования, разъясняющих, как применять язык, но не обучающих собственно программированию.

Для кого предназначена эта книга

Язык UML представляет интерес для любого специалиста, участвующего в процессе разработки, внедрения и сопровождения программного обеспечения. Данное руководство пользователя в первую очередь предназначено для разработчиков, создающих модели UML. В то же время оно будет полезно всем, кто читает эти модели в процессе анализа, создания, тестирования и выпуска версий программных систем. Хотя под такое определение подходит чуть ли не любой сотрудник организации, занимающейся разработкой программного обеспечения, книга *особенно* пригодится аналитикам и конечным пользователям, которые специфицируют требуемую структуру и поведение системы, архитекторам, которые проектируют системы в соответствии с этими требованиями, разработчикам, преобразующим проект в исполняемый код, специалистам по обеспечению качества, проверяющим и подтверждающим соответствие структуры и поведения системы заданным спецификациям, библиотекарям, которые создают каталоги компонентов, а также руководителям программ и проектов, которые борются с хаосом, осуществляют общее руководство, определяют направление работ и распределяют необходимые ресурсы.

Данная книга рассчитана на читателей, которые имеют хотя бы общее представление об объектно-ориентированных концепциях. Опыт работы с языками или методами объектно-ориентированного программирования не требуется, хотя и желателен.

Как работать с этой книгой

Тем, кто только начинает осваивать язык UML, лучше всего читать эту книгу последовательно. Особое внимание рекомендуется уделить второй главе, в которой излагается концептуальная модель языка. Содержание каждой главы опирается на материал предыдущей, что делает книгу удобной для последовательного чтения.

Опытные разработчики, желающие найти решение конкретных проблем, возникающих при моделировании, могут изучать это руководство в любой последовательности. Советуем обратить внимание на типичные приемы моделирования, представленные в каждой главе.

Организация книги и особенности изложения материала

Данное руководство пользователя содержит семь основных разделов:

- Часть I. Введение в процесс моделирования.
- Часть II. Основы структурного моделирования.
- Часть III. Расширенное структурное моделирование.
- Часть IV. Основы моделирования поведения.
- Часть V. Расширенное моделирование поведения.
- Часть VI. Моделирование архитектуры.
- Часть VII. Итоги.

Кроме того, в книгу включены два приложения: обзор применяемой в языке UML нотации и обзор технологии Rational Unified Process, а также глоссарий, содержащий наиболее распространенные термины.

Каждая глава посвящена рассмотрению какой-то конкретной возможности UML и, как правило, состоит из следующих четырех разделов:

1. Введение.
2. Термины и понятия.
3. Типичные приемы моделирования.
4. Советы и подсказки.

В начале каждой главы приводится список обсуждаемых в ней тем.

В разделе «Типичные приемы моделирования» содержатся примеры решения задач, часто возникающих при моделировании. Для облегчения работы с руководством каждая задача приводится под отдельным заголовком.

Комментарии и советы по теме главы приводятся в разделах «На заметку».

Язык UML имеет широкие семантические возможности. По этой причине описание одной особенности может пересекаться с описанием другой. В таких случаях слева приводятся ссылки на перекрестные темы.

Серый цвет ссылки используется для объяснения модели, которая, в отличие от объяснения, всегда изображается черным. Фрагменты кода выделяются `моноширинным шрифтом`.

Благодарности

Авторы хотят выразить благодарность Брюсу Дугласу (Bruce Douglass),

Перу Кроллу (Per Kroll) и Хоакину Миллеру (Joaquin Miller) за их помощь в создании книги.

Краткая история UML

Первым объектно-ориентированным языком принято считать Simula-67, разработанный Далем и Нигардом в Норвегии в 1967 году. Этот язык мало кто взял на вооружение, но его концепция во многом способствовала появлению других языков. Язык Smalltalk получил широкое распространение в начале 1980-х годов, а в конце того же десятилетия за ним последовали другие объектно-ориентированные языки, такие как Objective C, C++ и Eiffel. Объектно-ориентированные языки моделирования появились в 1980-х годах, когда исследователи, поставленные перед необходимостью учитывать новые возможности объектно-ориентированных языков программирования и требования, предъявляемые все более сложными приложениями, вынуждены были начать разработку различных альтернативных подходов к анализу и проектированию. В период с 1989 по 1994 год число объектно-ориентированных методов возросло с десяти более чем до пятидесяти. Тем не менее многие пользователи испытывали затруднения при выборе языка моделирования, который полностью отвечал бы их потребностям, что послужило причиной так называемой «войны методов». В результате появилось новое поколение методов, среди которых особое значение приобрели метод Буча, OOSE (Object-Oriented Software Engineering), разработанный Якобсоном, и OMT (Object Modeling Technique), разработанный Рамбо. Кроме того, следует упомянуть методы Fusion, Shlaer-Mellor и Coad-Yourdon. Каждый из этих методов можно было считать целостным и законченным, хотя любой из них имел не только сильные, но и слабые стороны. Выразительные возможности метода Буча были особенно важны на этапах проектирования и конструирования системы. OOSE был великолепно приспособлен для анализа и формулирования требований, а также для высокоуровневого проектирования. OMT оказался особенно полезным для анализа и разработки информационных систем.

Критическая масса новых идей начала накапливаться к середине 1990-х годов, когда Гради Буч (компания Rational Software Corporation), Джеймс Рамбо (General Electric), Ивар Якобсон (Objectory) и другие предприняли попытку объединить свои методы, уже получившие мировое признание как наиболее перспективные в области объектно-ориентированных методов. Являясь основными авторами методов Буча, OOSE и OMT, мы попытались создать новый, унифицированный язык моделирования и руководствовались при этом тремя соображениями. Во-первых, все три метода независимо от желания разработчиков уже развивались во встречном направлении. Разумно было продолжить это движение вместе, а не по отдельности, что помогло бы в будущем устранить нежелательные различия и, как следствие, неудобства для пользователей. Во-вторых, унифицировав

методы, проще было привнести стабильность на рынок инструментов объектно-ориентированного моделирования, что дало бы возможность положить в основу всех проектов единый зрелый язык, а создателям инструментальных средств позволило бы сосредоточиться на более продуктивной деятельности. Наконец, следовало полагать, что подобное сотрудничество приведет к усовершенствованию всех трех методов и обеспечит решение задач, для которых любой из них, взятый в отдельности, был не слишком пригоден.

Начав унификацию, мы поставили перед собой три главные цели:

1. Моделировать системы целиком, от концепции до исполняемых компонентов, с помощью объектно-ориентированных методов;
2. Решить проблему масштабируемости, которая присуща сложным, критически важным системам;
3. Создать язык моделирования, который может использоваться не только людьми, но и компьютерами.

Создание языка для объектно-ориентированного анализа и проектирования не слишком отличается от разработки языка программирования. Во-первых, требовалось ограничить задачу. Следует ли включать в язык возможность спецификации требований? Должен ли язык позволять визуальное программирование? Во-вторых, необходимо было найти точку равновесия между выразительной мощью и простотой. Слишком простой язык ограничил бы круг решаемых с его помощью задач, а слишком сложный мог ошеломить неискушенного разработчика. Кроме того, при объединении существующих методов следовало учитывать наличие продуктов, разработанных с их помощью. Внесение слишком большого числа изменений могло бы оттолкнуть уже имеющихся пользователей, а авторы, сопротивляясь развитию языка, потеряли бы возможность привлекать новых пользователей и делать язык более простым и удобным для применения. Создавая UML, мы старались найти оптимальное решение этих проблем.

Официальное создание UML началось в октябре 1994 года, когда Рамбо перешел в компанию Rational Software, где работал Буч. Первоначальной целью было объединение методов Буча и OMT. Первая пробная версия 0.8 Унифицированного метода (Unified Method), как его тогда называли, появилась в октябре 1995 года. Приблизительно в то же время в компанию Rational перешел Якобсон, и проект UML был расширен с целью включить в него OOSE. В результате наших совместных усилий в июне 1996 года вышла версия 0.9 языка UML. На протяжении всего года создатели занимались сбором отзывов от основных компаний, работающих в области создания программного обеспечения. За это время стало ясно, что большинство таких компаний сочло UML языком, имеющим стратегическое значение для их бизнеса.

В результате был создан консорциум UML, в который вошли организации, изъявившие желание предоставить ресурсы для работы, направленной на создание полной спецификации UML. Версия 1.0 языка появилась в результате совместных усилий компаний Digital Equipment Corporation, Hewlett-Packard, I-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational, Texas Instruments и Unisys. UML 1.0 оказался хорошо определенным, выразительным, мощным языком, применимым для решения большого количества разнообразных задач. В январе 1997 года он был предоставлен в качестве проекта стандартного языка моделирования консорциуму OMG (Object Management Group).

Между январем и июлем 1997 года консорциум UML расширился – в него вошли почти все компании, откликнувшиеся на призыв OMG, а именно: Andersen Consulting, Ericsson, Object Time Limited, Platinum Technology, Ptech, Reich Technologies, Softeam, Sterling Software и Taskon. Чтобы формализовать спецификации UML и координировать работу с другими группами, занимающимися стандартизацией, под руководством Криса Кобрина (Cris Kobryn) из компании MCI Systemhouse и Эда Эйхолта (Ed Eykholt) из Rational была организована рабочая группа. Пересмотренная версия UML (1.1) была представлена на рассмотрение OMG в июле 1997 года. В сентябре версия была утверждена на заседаниях группы по анализу и проектированию и Комитета по архитектуре OMG, а 14 ноября 1997 года была утверждена в качестве стандарта всеми участниками консорциума OMG.

В течение нескольких лет специальная рабочая группа OMG (OMG Revision Task Force) поддерживала продвижение проекта UML. Были созданы версии 1.3, 1.4 и 1.5. За 2000–2003 годы новая, расширенная группа участников проекта создала модернизированную версию UML 2.0. Эта версия рассматривалась в течение года рабочей группой Finalization Task Force (FTF) во главе с Брэном Сэликом (Bran Selic) из IBM, а в начале 2005 года члены OMG утвердили официальную версию UML 2.0. UML 2.0 – это последний релиз UML, который включает в себя большое количество дополнительных возможностей. Много изменений было сделано благодаря опыту использования предыдущих версий. Текущую спецификациюUML вы найдете на Web-сайте OMG www.omg.org.

UML – это результат работы множества специалистов и осмысления опыта их предыдущих разработок. Подсчитать источники, которые использовались при подготовке этого проекта, было бы огромным научным трудом. Но сложнее всего было бы установить все предшествующие наработки, в большей или меньшей степени оказавшие влияние на UML. С учетом всех научных исследований и рабочей практики можно сказать, что UML – это верхушка «айсберга», который вобрал в себя весь предыдущий опыт.

Часть I

Введение в процесс моделирования

Глава 1. Зачем мы моделируем

Глава 2. Введение в UML

Глава 3. Здравствуй, мир!

Глава 1. Зачем мы моделируем

В этой главе:

- Значение моделирования
- Четыре принципа моделирования
- Наиболее важные модели программных систем
- Объектно-ориентированное моделирование

Компания, занимающаяся производством программного обеспечения, может преуспевать только в том случае, если выпускаемая ею продукция отличается высоким качеством и разработана в соответствии с запросами пользователей. Фирма, которая способна выпускать такую продукцию своевременно и регулярно, при максимально полном и эффективном использовании всех имеющихся человеческих и материальных ресурсов будет стабильно процветать.

Из сказанного следует, что основным продуктом такой компании является первоклассное программное обеспечение, удовлетворяющее повседневные нужды пользователей. Все остальное – прекрасные документы, встречи на высшем уровне, великолепные лозунги и даже Пулитцеровская премия за идеальные строки исходного кода – вторично по сравнению с этой основной задачей.

К сожалению, во многих организациях путают понятия «вторичный» и «несущественный». Нельзя забывать, что для разработки эффективной программы, которая соответствует своему предполагаемому назначению, необходимо постоянно встречаться и работать с пользователем, чтобы выяснить реальные требования к вашей системе. Если вы хотите создать качественное программное обеспечение, вам необходимо разработать прочное архитектурное основание проекта, открытое к возможным усовершенствованиям. Для быстрой и эффективной разработки программного продукта с минимальным браком требуется привлечь рабочую силу, выбрать правильные инструменты и определить верное направление работы. Чтобы справиться с поставленной задачей, принимая во внимание затраты на жизненный цикл системы, необходимо, чтобы процесс разработки приложения был тщательно продуман и мог

быть адаптирован к изменяющимся потребностям вашего бизнеса и технологии.

Центральным элементом деятельности, ведущей к созданию первоклассного программного обеспечения, является моделирование. Модели позволяют нам наглядно продемонстрировать желаемую структуру и поведение системы. Они также необходимы для визуализации и управления ее архитектурой. Модели помогают добиться лучшего понимания создаваемой нами системы, что зачастую приводит к ее упрощению и обеспечивает возможность повторного использования. Наконец, модели нужны для минимизации риска.

Значение моделирования

Если вы хотите соорудить собачью будку, то можете приступить к работе, имея в наличии лишь кучу досок, горсть гвоздей, молоток, плоскогубцы и рулетку. Несколько часов работы после небольшого предварительного планирования – и вы, надо полагать, сколотите вполне приемлемую будку, причем, скорее всего, без посторонней помощи. Если будка получится достаточно большой и не будет сильно протекать, собака останется довольна. В крайнем случае никогда не поздно начать все сначала – или приобрести менее капризного пса.

Если вам надо построить дом для своей семьи, вы, конечно, можете воспользоваться тем же набором инструментов, но времени на это уйдет значительно больше, и ваши домочадцы, надо полагать, окажутся более требовательными, чем собака. Если у вас нет особого опыта в области строительства, лучше тщательно все продумать перед тем, как забить первый гвоздь. Стоит, по меньшей мере, сделать хотя бы несколько эскизов будущей постройки. Без сомнения, вам нужно качественное жилье, удовлетворяющее запросам вашей семьи и не нарушающее местных строительных норм и правил, – а значит, придется сделать кое-какие чертежи с учетом назначения каждой комнаты и таких деталей, как освещение, отопление и водопровод. На основании этих планов вы сможете правильно рассчитать необходимое для работы время и выбрать подходящие стройматериалы. В принципе можно построить дом и самому, но гораздо выгоднее прибегнуть к помощи других людей, нанимая их для выполнения ключевых работ, или хотя бы купить готовые детали. Коль скоро вы следуете плану и укладываетесь в смету, ваша семья будет довольна. Если же что-то не сладится, вряд ли стоит менять семью – лучше своевременно учесть пожелания родственников.

Задавшись целью построить небоскреб для офиса, было бы совсем неразумно браться за дело, имея в распоряжении груду досок и молоток. Поскольку в этом случае, скорее всего, будут привлекаться

многочисленные капиталовложения, инвесторы потребуют, чтобы вы учли их пожелания относительно размера, стиля и формы строения. Кстати, они могут изменить свое решение уже после того, как вы приступите к строительству. Так как любой просчет вам обойдется слишком дорого, значение планирования многократно возрастает. По-видимому, вы окажетесь членом большого коллектива проектировщиков и застройщиков, и для успешного взаимодействия вам потребуется выполнять множество разнообразных чертежей и моделей здания.

Подобрав нужных людей, подходящие инструменты и приступив к реализации намеченного плана, вы с большой вероятностью сумеете построить здание, удовлетворяющее всем требованиям будущих обитателей. Если вы и дальше собираетесь возводить дома, то придется искать компромисс между пожеланиями заказчика и возможностями современной технологии строительства, а к рабочему коллективу относиться с заботой, не рискуя людьми и не требуя работы, отнимающей всех сил без остатка.

Хотя это и кажется комичным, многие компании, разрабатывающие программное обеспечение, хотят создать небоскреб, в то время как их подход к делу очень напоминает строительство собачьей будки.

Конечно, иногда вам может просто повезти. Если нужные люди оказались в нужном месте в нужное время и расположение планет благоприятствует новым начинаниям, то, возможно, вам посчастливится создать продукт, который подойдет потребителям. Но, скорее всего, нанять подходящих работников не удастся (все они уже заняты), благоприятный момент вы упустите (вчера было бы лучше), а расположение планет будет крайне неблагоприятным (этот факт вообще неподвластен вашему контролю). Сталкиваясь с возрастающими требованиями к скорости разработки программного обеспечения, коллективы разработчиков часто ограничиваются тем единственным, что они по-настоящему умеют делать – написанием новых строк кода. Героические усилия, затрачиваемые на программирование, стали легендой в этой отрасли, и кажется, что единственным ответом на трудности в разработке программного обеспечения послужит еще более интенсивная работа. Однако написанный код вовсе не обязательно поможет решить проблему, а проект может быть настолько грандиозным, что даже увеличение рабочего дня на несколько часов окажется недостаточным для его успешного завершения.

Если вы действительно хотите создать программный продукт, по масштабности замысла сопоставимый с жилым домом или небоскребом, то ваша задача не сводится к написанию большого объема кода. На самом деле проблема в том, чтобы написать *правильный*

код *минимального* объема. При таком подходе разработка качественного программного обеспечения сводится к вопросам выбора архитектуры, подходящего инструментария и средств управления процессом. Кстати, нужно иметь в виду, что многие проекты, задуманные «по принципу будки», быстро вырастают до размеров небоскреба, становясь жертвой собственного успеха. Если такой рост не был учтен в архитектуре приложения, технологическом процессе или при выборе инструментария, то неизбежно наступает время, когда выросшая до размеров огромного дома будка обрушится под тяжестью собственного веса. Но если разрушится будка, то это лишь разозлит вашу собаку, а если рухнет небоскреб, это затронет материальные интересы его арендаторов.

Неудачные проекты терпят крах в силу самых разных причин, а вот успешные, как правило, имеют много общего. Хотя успех программного проекта обеспечивается множеством разных составляющих, одной из главных является применение моделирования.

Моделирование – это устоявшаяся и повсеместно принятая инженерная методика. Мы строим архитектурные модели зданий, чтобы помочь их будущим обитателям во всех деталях представить готовый объект. Иногда применяют даже математическое моделирование зданий, чтобы учесть влияние сильного ветра или землетрясения.

Моделирование применяется не только в строительстве. Вряд ли вы сумеете наладить выпуск новых самолетов или автомобилей, не испытав свой проект на моделях: от компьютерных моделей и чертежей до физических моделей в аэродинамической трубе, а затем и полномасштабных прототипов. Электрические приборы от микропроцессоров до телефонных коммутаторов также требуют моделирования для лучшего понимания системы и организации общения разработчиков друг с другом. Даже в кинематографии успех фильма невозможен без предварительно написанного сценария (тоже своеобразная форма модели!) В социологии, экономике или менеджменте мы также прибегаем к моделированию, которое позволяет проверить наши теории и испытать новые идеи с минимальным риском и затратами.

Итак, что же такое модель? Попросту говоря, *модель – это упрощенное представление реальности.*

Модель – это чертеж системы: в нее может входить как детальный план, так и более абстрактное представление системы «с высоты птичьего полета». Хорошая модель всегда включает элементы, которые существенно влияют на результат, и не включает те, которые малозначимы на данном уровне абстракции. Каждая система может быть описана с разных точек зрения, для чего используются разные модели, каждая из которых, следовательно, является семантически

замкнутой абстракцией системы. Модель может быть структурной, подчеркивающей организацию системы, или же поведенческой, отражающей ее динамику.

Зачем мы моделируем? Для этого есть одна фундаментальная причина.

Мы строим модель для того, чтобы лучше понимать разрабатываемую систему.

Моделирование позволяет решить четыре различные задачи (см. главу 2):

1. Визуализировать систему в ее текущем или желательном для нас состоянии;
2. Описать структуру или поведение системы;
3. Получить шаблон, позволяющий сконструировать систему;
4. Документировать принимаемые решения, используя полученные модели.

Моделирование предназначено не только для создания больших систем. Даже программный эквивалент собачьей будки выиграет от его применения. Чем больше и сложнее система, тем большее значение приобретает моделирование при ее разработке. Дело в том, что *мы строим модели сложных систем, поскольку иначе такие системы невозможно воспринять как единое целое.*

Человеческое восприятие сложных сущностей ограничено. Моделируя, мы сужаем проблему, акцентируем внимание в каждый данный момент только на одном ее аспекте. По сути, этот подход есть не что иное, как принцип «разделяй и властвуй», который Эдсгер Дейкстра (Edsger Dijkstra) провозгласил много лет назад: сложную задачу легче решить, если разделить ее на несколько меньших. Кроме того, моделирование усиливает возможности человеческого интеллекта, поскольку правильно выбранная модель позволяет создавать проекты на более высоких уровнях абстракции.

Сказать, что моделирование имеет смысл, – еще не значит, что оно абсолютно необходимо. И действительно, многие исследования показывают, что в большинстве компаний, разрабатывающих программное обеспечение, моделирование применяется редко или же не применяется вообще. Чем проще проект, тем менее вероятно, что в нем будет использовано формальное моделирование.

Ключевое слово здесь – «формальное». На практике даже при реализации простейшего проекта разработчики в той или иной мере применяют моделирование, хотя бы неформально. Для визуализации части системы ее проектировщик может нарисовать что-то на доске или клочке бумаги. Коллектив разработчиков может применять CRC-карточки, чтобы проработать сценарий или конструкцию механизма. Ничего плохого в таких моделях нет. Если они

работают, то их существование оправдано. Но такие неформальные модели часто создаются для однократного применения и не обеспечивают общего языка, который был бы понятен другим участникам проекта. В строительстве существует общий, понятный для всех язык чертежей, имеются общие языки в электротехнике и математическом моделировании. Организация разработчиков также может извлечь выгоду из применения общего языка моделирования программного обеспечения.

От моделирования может выиграть любой проект. Даже при создании одноразовых программ, когда зачастую бывает полезно выбрать неподходящий код из-за преимущества в скорости разработки, которое дают языки визуального программирования, моделирование поможет коллективу разработчиков лучше представить план системы, а значит, выполнить проект быстрее и создать именно то, что подразумевал первоначальный замысел. Чем сложнее проект, тем более вероятно, что из-за отсутствия моделирования он свернется раньше времени – или будет создано не то, что нужно. Все полезные и интересные системы с течением времени обычно усложняются. Пренебрегая моделированием в самом начале создания системы, вы, возможно, серьезно пожалеете об этом, когда будет уже слишком поздно.

Принципы моделирования

Моделирование имеет богатую историю во всех инженерных дисциплинах. Длительный опыт его использования позволил сформулировать четыре основных принципа.

Во-первых, *выбор модели оказывает определяющее влияние на подход к решению проблемы и на то, как будет выглядеть это решение*.

Иначе говоря, подходите к выбору модели вдумчиво. Правильно выбранная модель высветит самые коварные проблемы разработки и позволит проникнуть в самую суть задачи, что при ином подходе было бы просто невозможно. Неправильная модель заведет вас в тупик, поскольку внимание будет заостряться на несущественных вопросах.

Давайте отвлечемся на минутку от обсуждения программного обеспечения и предположим, что вам нужно решить задачу из области квантовой физики. Такие проблемы, как взаимодействие фотонов в пространстве-времени, могут потребовать чрезвычайно сложных математических расчетов. Но стоит изменить модель, забыв о математическом анализе, и вы тут же получите легко интерпретируемый результат. В данном случае речь идет о диаграммах Феймана, позволяющих графически представить чрезвычайно сложные проблемы. Рассмотрим еще одну область, где необходимо моделирование.

Допустим, вы проектируете здание и хотите знать, как оно будет себя вести при сильном ветре. Построив макет и испытав его в аэродинамической трубе, вы узнаете много интересного по этому поводу, хотя уменьшенная копия будет себя вести не так, как полно ем математической модели даст вам дополнительную информацию и, возможно, позволит проработать больше различных сценариев, чем при работе с физическим макетом. Тщательно изучив поведение здания на этих моделях, вы будете гораздо больше уверены в том, что смоделированная система будет вести себя в реальных условиях так, как было задумано.

Возвращаясь к проблеме создания программного обеспечения, можно сказать, что ваш взгляд на мир существенно зависит от выбираемой вами модели. Если вы смотрите на систему глазами разработчика баз данных, то основное внимание будете уделять моделям «сущность–связь», где поведение инкапсулировано в триггерах и хранимых процедурах. Аналитик, использующий структурный подход, скорее всего, создал бы модель, в центре которой находятся алгоритмы и передача данных от одного процесса к другому. Результатом труда разработчика, пользующегося объектно-ориентированным методом, будет система, архитектура которой основана на множестве классов и образцах взаимодействия, определяющих, как эти классы действуют совместно. Любой из этих вариантов может оказаться подходящим для данного приложения и методики разработки, хотя опыт подсказывает, что объектно-ориентированная точка зрения более эффективна при создании гибких архитектур, даже если система должна будет работать с большими базами данных или производить сложные математические расчеты. При этом надо учитывать, что различные точки зрения на мир приводят к созданию различных систем, со своими преимуществами и недостатками.

Второй принцип формулируется так: *каждая модель может быть представлена с различной степенью точности.*

При строительстве небоскреба может возникнуть необходимость показать его с высоты птичьего полета, например, чтобы с проектом могли ознакомиться инвесторы. В других случаях, наоборот, требуется детальное описание – допустим, чтобы показать какой-нибудь сложный изгиб трубы или необычный элемент конструкции.

То же происходит и при моделировании программного обеспечения. Иногда простая и быстро созданная модель программного интерфейса – самый подходящий вариант. В других случаях приходится работать на уровне битов (например, когда вы специфицируете межсистемные интерфейсы или боретесь с узкими местами в сети). В любом случае лучшей моделью будет та, которая позволяет выбрать уровень детализации в зависимости от того, кто и с какой целью на нее смотрит. Для аналитика или конечного пользователя

наибольший интерес представляет вопрос «что», а для разработчика – вопрос «как». В обоих случаях необходима возможность рассматривать систему на разных уровнях детализации в разное время.

Третий принцип: *лучшие модели – те, что ближе к реальности*.

Физическая модель здания, которая ведет себя не так, как изготовленная из реальных материалов, имеет лишь ограниченную ценность. Математическая модель самолета, для которой предполагаются идеальные условия работы и безупречная сборка, может и не обладать некоторыми характеристиками, присущими настоящему изделию, что в ряде случаев приводит к фатальным последствиям. Лучше всего, если ваши модели будут во всем соотноситься с реальностью, а там, где связь ослабевает, должно быть понятно, в чем заключается различие и что из этого следует. Поскольку модель всегда упрощает реальность, задача в том, чтобы это упрощение не повлекло за собой какие-то существенные потери.

Возвращаясь к программному обеспечению, можно сказать, что «ахиллесова пята» структурного анализа – несоответствие принятой в нем модели и модели системного проекта. Если этот разрыв не будет устранен, то поведение созданной системы с течением времени будет все больше и больше отличаться от задуманного. При объектно-ориентированном подходе можно объединить все почти независимые представления системы в единое семантическое целое.

Четвертый принцип заключается в том, что *нельзя ограничиваться созданием только одной модели. Наилучший подход при разработке любой нетривиальной системы – использовать совокупность нескольких моделей, почти независимых друг от друга*.

Если вы конструируете здание, никакой отдельный комплект чертежей не поможет вам прояснить все детали до конца. Понадобятся как минимум поэтажные планы, виды в разрезе, схемы электропроводки, центрального отопления и водопровода.

В формулировке четвертого принципа ключевое определение – «почти независимые». Оно означает, что модели могут создаваться и изучаться по отдельности, но вместе с тем остаются взаимосвязанными. Например, можно изучать только схемы электропроводки проектируемого здания, но при этом наложить их на поэтажный план пола и даже рассмотреть совместно с прокладкой труб на схеме водоснабжения.

Такой подход верен и в объектно-ориентированных программных системах. Для понимания архитектуры подобной системы требуется несколько взаимодополняющих представлений: представление с точки зрения вариантов использования (чтобы выявить требования к системе), с точки зрения проектирования (чтобы построить словарь предметной области и области решения), с точки зрения взаимодействий (чтобы смоделировать взаимодействия

между частями системы, системой в целом и средой ее функционирования), с точки зрения реализации (позволяющее рассмотреть физическую реализацию системы) и с точки зрения размещения (помогающее сосредоточиться на вопросах системного проектирования). Каждое из перечисленных представлений имеет множество структурных и поведенческих аспектов, которые в своей совокупности составляют детальный чертеж программной системы.

Пять представлений архитектуры обсуждаются в главе 2.

В зависимости от природы системы некоторые модели могут быть важнее других. Так, при создании систем для обработки больших объемов данных более важны статические модели. В приложениях, ориентированных на интерактивную работу пользователя, на первый план выходят статические и динамические представления вариантов использования. В системах реального времени наиболее существенными будут представления с точки зрения динамических процессов. Наконец, в распределенных системах, таких как Web-приложения, основное внимание нужно уделять моделям реализации и размещения.

Объектное моделирование

Инженеры-строители создают множество видов моделей. Чаще всего это структурные модели, позволяющие визуализировать и специфицировать части системы и то, как они соотносятся друг с другом. Иногда создаются также динамические модели – например, если требуется изучить поведение конструкции при землетрясении. Эти два типа различаются по организации и по тому, на что в первую очередь обращается внимание при проектировании.

При разработке программного обеспечения тоже существует несколько подходов к моделированию. Важнейшие из них – алгоритмический и объектно-ориентированный.

Алгоритмический метод представляет традиционный подход к созданию программного обеспечения. Основным строительным блоком является процедура или функция, а внимание уделяется прежде всего вопросам передачи управления и вопросам декомпозиции больших алгоритмов на меньшие. Ничего плохого в этом нет, если не считать того, что системы не слишком легко адаптируются. При изменении требований или увеличении размера приложения (что происходит нередко) сопровождать их становится сложнее.

Наиболее современный подход к разработке программного обеспечения – объектно-ориентированный. Здесь в качестве основного строительного блока выступает объект или класс. В самом общем смысле *объект* – это сущность, обычно извлекаемая из словаря предметной области или решения, а *класс* – описание множества однотипных объектов. Каждый объект обладает идентичностью

(его можно поименовать или как-то по-другому отличить от прочих объектов), состоянием (обычно с объектом связаны некоторые данные) и поведением (с ним можно что-то делать или он сам может что-то делать с другими объектами).

В качестве примера давайте рассмотрим простую трехуровневую архитектуру биллинговой системы, состоящую из интерфейса пользователя, бизнес-логики (промежуточного слоя) и базы данных. Пользовательский интерфейс состоит из конкретных объектов: кнопок, меню и диалоговых окон. База данных также состоит из конкретных объектов, а именно таблиц, представляющих сущности предметной области: клиентов, продукты и заказы. Программы промежуточного слоя включают такие объекты, как транзакции и бизнес-правила, а кроме того, более абстрактные представления сущностей предметной области (клиентов, продуктов и заказов).

Объектно-ориентированный подход к разработке программного обеспечения сейчас наиболее широко используется потому, что он продемонстрировал свою полезность при построении систем любого размера и сложности в самых разных областях. Кроме того, большинство современных языков программирования, инструментальных средств и операционных систем являются в той или иной мере объектно-ориентированными, а это дает веские основания судить о мире в терминах объектов. Объектно-ориентированные методы разработки легли в основу идеологии сборки систем из отдельных компонентов (в качестве примеров можно назвать такие технологии, как J2EE и .NET).

Если вы приняли объектно-ориентированный взгляд на мир, вам придется ответить на ряд вопросов. Какая структура должна быть у хорошей объектно-ориентированной архитектуры? Какие артефакты должны быть созданы в процессе работы над проектом? Кто должен создавать их? И наконец, как оценить результат?

Компоненты обсуждаются в главе 2.

Визуализация, специфицирование, конструирование и документирование объектно-ориентированных систем – это и есть назначение языка UML.