Содержание

От редакторов

Добрый день! Спасибо, что вы открыли эту книгу, даже если взяли с полки магазина просто посмотреть. Нам приятно в любом случае. Для нас удача и честь принять участие в переводе такой легендарной книги, чтобы вы могли прочесть ее на родном языке.

Что же делает ее легендарной? Вы наверняка заметили, что это перевод *второго* издания, которое вышло уже почти 5 лет назад, а первое – более 20 лет назад. Оставаться востребованным столько лет в стремительно меняющемся мире информационных технологий и языков программирования – достижение само по себе.

Авторы книги – легенды мира компьютерных наук и информатики, авторы фундаментальных работ на протяжении последних 30 лет. Они же являются одними из пионеров систематического, научного подхода к *преподаванию* программирования и информатики. Как результат, данная книга является отражением глубокого понимания как самого предмета, так и методики его преподавания, сформировавшегося у авторов за десятилетия практики.

Другой фактор долголетия этого учебника – то, что он *живой*. Книга продолжает развиваться даже после публикации: в неё постоянно вносят правки и уточнения, исправляют ошибки, она используется в качестве основы для курсов в университетах и на онлайн-площадках.

В результате получился один из лучших в мире учебников по программированию для новичков, который увлекательно читать, который никогда не перегружает лишними деталями, всегда ясно указывает направление развития и ненавязчиво прививает правила «хорошего тона».

Авторы справедливо отмечают, что это учебник по чему-то большему, чем просто программирование. В первую очередь он учит тому, что в последнее время принято называть «computational thinking», «алгоритмическим (или вычислительным) мышлением». Этим стилем мышления пользуются не только программисты, но и повара, учителя, спортсмены, врачи и многие, многие другие.

Авторы книги скромничают, когда говорят, что книга ориентирована на начинающих. Профессионалы встретят в ней как отсылки к фундаментальным понятиям и методам компьютерных наук, так и эффективные методы анализа повседневных задач. Поэтому от всего сердца советуем вам купить эту книгу! Мы от этого богаче не станем, но верим, что вас она способна обогатить интеллектуально, творчески и профессионально, даже если разработка программного обеспечения – не ваша основная деятельность.

С уважением от редакторов,

Павел Борисович Иванов,
Александр Дмитриевич Чичигин,
Юрий Алексеевич Сыровецкий,
Сергей Викторович Бронников

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и The MIT Press очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Вступление

Многие современные профессии требуют умения программировать в той или иной форме. Бухгалтеры программируют электронные таблицы; музыканты программируют синтезаторы; писатели программируют текстовые процессоры; а веб-дизайнеры программируют таблицы стилей. Когда мы писали эти слова для первого издания книги (1995–2000), читатели могли счесть их футуристическими, однако к настоящему времени умение программировать стало обязательным, и появились многочисленные книги, онлайн-курсы и предметы в общеобразовательной школе, которые удовлетворяют эту потребность и улучшают шансы людей на трудоустройство.

Типичный курс программирования учит подходу «Пробуй, пока не заработает». Добившись нужного результата, учащийся восклицает: «Работает!» – и идет дальше. К сожалению, эта фраза также является самой короткой небылицей в информатике и многим людям стоила многих часов их жизни. Эта книга, напротив, фокусируется на навыках *хорошего программирования* и адресована всем – и профессиональным программистам, и любителям.

Под «хорошим программированием» мы подразумеваем подход к созданию программного обеспечения, который изначально опирается на системное мышление, планирование и понимание на каждом этапе и на каждом шаге. Чтобы подчеркнуть это, мы говорим о системном *проектировании программ* и системно *спроектированных программах*. Что особенно важно, последнее словосочетание ясно выражает требование к желаемой функциональности. Хорошее программирование также удовлетворяет эстетическое чувство выполненного долга; хорошая программа по своей элегантности сравнима с хорошими стихами или черно-белыми фотографиями ушедшей эпохи. Проще говоря, программирование отличается от хорошего программирования как наброски карандашом на салфетке, сделанные в закусочной, от картин маслом в музее.

Нет, эта книга не превратит вас в мастера живописи. Но мы не стали бы тратить пятнадцать лет на подготовку данного издания, если бы не верили, что

каждый может разрабатывать программы

и

каждый может испытывать удовлетворение от творческого процесса.

Более того, мы утверждаем, что

*проектирование программ – но **не программирование** –*
в традиционном для Запада высшем образовании должно стоять
рядом с математикой и лингвистикой.

Студент, изучающий проектирование, который никогда больше не коснется программ, все равно приобретет универсально полезные

навыки решения задач, приобретет опыт творческой деятельности и научится ценить новую форму эстетики. Остальная часть этого вступления подробно объясняет, что мы имеем в виду под «системным проектированием», кому и чем это выгодно и как мы обучаем всему этому.

Системное проектирование программ

Программа взаимодействует с людьми, которых называют *пользователями*, и другими программами, которые могут быть *серверами* или *клиентами*. Соответственно, любая более или менее полная программа состоит из множества строительных блоков, одни из которых обрабатывают ввод, другие производят вывод, а третьи соединяют первые со вторыми. В качестве фундаментальных строительных блоков мы предпочитаем использовать функции, потому что все мы хорошо знакомы с функциями по курсу школьной математики и потому что простейшие программы являются именно такими функциями. Главное – выяснить, какие функции необходимы, как их соединить между собой и как их сконструировать из основных ингредиентов.

Мы черпали вдохновение в методе, предложенном Майклом Джексоном (Michael Jackson) для создания программ на языке COBOL, а также в беседах с Дэниелом Фридманом (Daniel Friedman) о рекурсии, Робертом Харпером (Robert Harper) о теории типов и Дэниелом Джексоном (Daniel Jackson) о проектировании программного обеспечения.

В этом контексте «системное проектирование программ» означает сочетание двух составляющих: рецептов проектирования и итеративного уточнения. Рецепты проектирования – это изобретение авторов, обеспечивающее возможность итеративного уточнения.

Рецепты проектирования применимы как к целым программам, так и к отдельным функциям. В этой книге есть всего два рецепта для целых программ: один для программ с графическим пользовательским интерфейсом (Graphical User Interface, GUI) и другой для неинтерактивных программ. Рецепты проектирования функций, напротив, намного разнообразнее: для данных атомарных типов, таких как числа; для перечислений разных видов данных; для данных, которые фиксированным образом объединяют другие данные; для конечных, но произвольно больших данных; и так далее.

Преподавателям. *Попросите учащихся скопировать рецепт 1 на картонную карточку. Когда у кого-то из них возникнет проблема, попросите их предъявить карточку и указать шаг, на котором они застряли.*

Рецепты проектирования для функций объединяются общим **процессом проектирования**. В списке в рецепте 1 перечислены шесть основных шагов этого процесса. Название каждого шага сообщает ожидаемый результат(ы), а «команды» определяют ключевые действия. Центральную роль в этом процессе играют примеры. Для представления данных, выбранного на шаге 1, примеры иллюстрируют, как фактическая информация кодируется в данные и как данные интерпретируются в информацию. В шаге 3 говорится, что человек, решающий задачу, должен проработать конкрет-

ные сценарии, чтобы понять, что должна вычислить функция в каждом конкретном примере. Это понимание используется на шаге 5, когда наступает время определения функции. Наконец, шаг 6 требует, чтобы примеры были преобразованы в автоматизированные тесты, проверяющие правильность работы функции в некоторых случаях. Запуск функции на реальных данных может выявить другие расхождения между ожиданиями и результатами.

Рецепт 1. ***Базовый рецепт проектирования функций***

1. **Анализ задачи и определение данных.** Определите, какая информация и как должна быть представлена в выбранном языке программирования. Сформулируйте определения данных и проиллюстрируйте их примерами.
2. **Сигнатура, описание назначения, заголовок.** Укажите, какие данные функция принимает и выдает. Сформулируйте короткий ответ на вопрос: «что вычисляет функция?» Определите заглушку с соответствующей сигнатурой.
3. **Примеры использования функции.** Представьте примеры, иллюстрирующие назначение функции.
4. **Создание макета функции.** Используя определения данных, напишите набросок функции.
5. **Определение функции.** Заполните недостающие части в макете функции. Используйте определение назначения и примеры.
6. **Тестирование.** Переформулируйте примеры в тесты и убедитесь, что функция успешно выполняет их все. Это поможет обнаружить вкравшиеся ошибки. Кроме того, тесты дополнят примеры и помогут другим понять определение функции, если в этом возникнет необходимость, а она всегда возникает в любой серьезной программе.

Преподавателям. *Наиболее важные вопросы возникают на шагах 4 и 5. Попросите учащихся записать эти вопросы своими словами на обратной стороне картонных карточек.*

На каждом шаге процесса проектирования возникают вопросы. На некоторых из них, например на шаге создания примеров использования функции или на шаге создания макета, вопросы могут затрагивать определение данных. Ответы на них почти автоматически создают промежуточный продукт. Затраты на эти подготовительные шаги окупаются, когда приходит время сделать творческий шаг – завершить определение функции, потому что оказывают необходимую помощь почти во всех случаях.

Необычность этого подхода заключается в создании новичками промежуточного продукта. Когда новичок застрянет на каком-то шаге, эксперт или преподаватель сможет проверить созданные промежуточные продукты, задать наводящие вопросы, характерные для процесса проектирования, и тем самым помочь новичку преодолеть затруднение. Этот аспект самообразования является коренным отличием проектирования программ от программирования.

Итеративное уточнение решает проблему сложности и многогранности задач. Сделать все и сразу практически невозможно. Для решения этой проблемы специалисты по информатике заимствовали метод итеративного уточнения из естественных наук. Согласно методу итеративного уточнения рекомендуется сначала отбросить все несущественные детали и найти решение основной задачи. Затем шаг уточнения добавляет одну из отброшенных деталей, и расширенная задача решается повторно с максимальным использованием существующего решения. Повторение этих шагов уточнения и поиска решения в конечном итоге приводит к полному решению.

В этом смысле программист является своего рода ученым. Ученые создают приблизительные модели идеализированной версии мира, чтобы получить возможность делать прогнозы. Пока прогнозы сбываются, модель используется как есть; когда прогнозы начинают отличаться от реальности, ученые пересматривают свои модели, стремясь уменьшить расхождения. Точно так же, когда программист получает задание, он создает первую модель, превращает ее в код, оценивает с помощью пользователей и итеративно уточняет модель, пока поведение программы не будет достаточно точно соответствовать желаемому.

В этой книге мы раскрываем итеративное уточнение с двух сторон. Поскольку проектирование методом итеративного уточнения становится особенно полезным при проектировании сложных программ, книга подробно знакомит с этой техникой в четвертой части, когда рассматриваемые задачи достигают определенной степени сложности. Кроме того, в первых трех частях итеративное уточнение используется для формулирования все более сложных вариантов одной и той же задачи. То есть в одной главе мы берем базовую задачу, решаем еще в одной главе, а в следующей главе ставим аналогичную задачу, но с дополнительными деталями, отражающими новые, свежие, только что введенные понятия.

DrRacket и учебные языки

Чтобы научиться проектировать программы, нужно постоянно практиковаться. Как нельзя стать пианистом, не играя на пианино, так же нельзя стать разработчиком программ, не создавая программ и не доводя их до корректного поведения. По этой причине наша книга сопровождается некоторой программной поддержкой: языком, на котором можно писать программы, и *средой разработки программ*, в которой программы редактируются как текстовые документы и с помощью которой можно запускать программы.

Многие, встречавшиеся нам и говорившие, что хотели бы научиться программированию, спрашивали, *какой язык программирования* им следует выучить. Учитывая рекламную шумиху, развернутую вокруг некоторых языков программирования, такой вопрос не вызывает удивления. Но

Преподавателям. *На курсах, предназначенных для опытных разработчиков, можно использовать другой подходящий язык.*

проблема в том, что он совершенно неуместен. Обучение программированию на языке, модном в настоящее время, часто приводит обучающихся к неудачам. Мода в этом мире очень недолговечна. Типичная книга «Короткий курс программирования на X» не может научить принципам, которые будут перенесены на следующий модный язык. Хуже того, сам язык часто отвлекает от приобретения передаваемых навыков на уровне формулирования решений и на уровне исправления ошибок.

Обучение проектированию программ, напротив, заключается прежде всего в изучении принципов и приобретении передаваемых навыков. Идеальный язык программирования должен поддерживать обе эти цели, чего нельзя сказать ни об одном стандартном промышленном языке. Ключевая проблема в том, что новички совершают ошибки еще до того, как более или менее существенно овладевают языком, однако языки программирования диагностируют эти ошибки, как если бы программист уже знал все его тонкости. В результате сообщения об ошибках часто ставят новичков в тупик.

Преподавателям. *Вы можете объяснить, что BSL – это школьная алгебра с дополнительными формами данных и множеством предопределенных функций для работы с ними.*

Наше решение: начать со знакомства с нашим собственным специализированным языком обучения, который мы назвали «язык для начинающих» (Beginning Student Language, BSL). По сути, это почти тот же «иностранный» язык, который изучается в школьном курсе математики. Он включает обозначения для определений функций, применения функций и условных выражений. Кроме того, он допускает вложенность выражений. То есть этот язык настолько мал, что диагностика ошибок в терминах всего языка доступна читателям, у которых нет никаких знаний, кроме начального курса математики.

Учащийся, овладевший принципами структурного проектирования, может затем перейти к «языку для учащихся промежуточной сложности» (Intermediate Student Language, ISL) и другим продвинутым диалектам с групповым названием `*SL`. В книге эти диалекты используются для обучения принципам абстракции и рекурсии. Мы твердо уверены, что использование такой последовательности обучающих языков позволяет читателям подготовить себя к созданию программ на широком спектре профессиональных языков программирования (JavaScript, Python, Ruby, Java и др.).

ПРИМЕЧАНИЕ. Обучающие языки реализованы на *Racket*, языке программирования для создания языков программирования. Racket ускользнул из лаборатории в реальный мир и постепенно стал применяться для решения самых разных задач, от создания игр до реализации управления массивами телескопов. Обучающие языки заимствуют некоторые элементы из языка Racket, но эта книга **не** учит программированию на Racket. Однако учащийся, прочитавший эту книгу, с легкостью сможет начать программировать на Racket. **КОНЕЦ.**

Выбирая среду программирования, мы оказываемся в такой же плохой ситуации, как при выборе языка программирования. Среда

программирования для профессионалов подобна кабине современного реактивного самолета. Она имеет множество элементов управления и дисплеев, непостижимых для любого, кто впервые запускает такое программное приложение. Начинающим программистам нужен эквивалент двухместного одномоторного поршневого самолета, на котором они могут практиковать базовые навыки. Поэтому мы создали среду программирования DrRacket для новичков.

DrRacket поддерживает очень увлекательное обучение с обратной связью с использованием всего двух простых интерактивных панелей: области определений, содержащей определения функций, и области взаимодействия, которая позволяет программисту запросить вычисление выражений, связанных с определениями. В этом контексте исследовать сценарии «что, если» так же легко, как в приложении для работы с электронными таблицами. Экспериментирование можно начать при первом же контакте, используя обычные примеры в стиле калькулятора и быстро переходя к вычислениям с изображениями, словами и другими формами данных.

Интерактивная среда разработки программ, такая как DrRacket, упрощает процесс обучения. Во-первых, она позволяет начинающим программистам напрямую манипулировать данными. Поскольку нет никаких средств для чтения входной информации из файлов или устройств хранения, новичкам не нужно тратить драгоценное время на выяснение особенностей их использования. Во-вторых, среда разработки четко отделяет данные и операции с ними от ввода и вывода информации из «реального мира». В настоящее время это разделение считается настолько фундаментальным для системного проектирования программного обеспечения, что получило собственное название: *архитектура модель-представление-контроллер* (Model-View-Controller, MVC). Работая в DrRacket, начинающие программисты естественным путем знакомятся с этой фундаментальной идеей программной инженерии.

Применение навыков

Приобретенные в процессе обучения навыки проектирования программ находят системное применение в двух направлениях: в программировании вообще и в программировании электронных таблиц, синтезаторов, таблиц стилей и даже текстовых процессоров в частности. Как показывают наши наблюдения, процесс проектирования, представленный в рецепте 1, легко перенести практически на любой язык программирования и можно использовать для разработки как коротких, насчитывающих десяток строк, так и длинных программ, состоящих из десятков тысяч строк кода. Конечно, необходимо какое-то время, чтобы осмыслить и адаптировать процесс проектирования ко всему спектру языков и к разным масштабам программ; но как

только он станет второй натурой, его применение становится естественным и начинает приносить выгоду.

Обучение проектированию программ также означает обретение двух универсальных навыков. Проектирование программ, безусловно, дает те же аналитические навыки, что и математика, особенно алгебра и геометрия. Но, в отличие от математики, работа с программами – это активный подход к обучению. Процесс создания программного обеспечения включает немедленную обратную связь и тем самым способствует исследованиям, экспериментам и самооценке. Результатом, как правило, являются интерактивные продукты, создание которых дает более мощное чувство удовлетворенности, чем решение упражнений в учебниках.

Проектирование программ тренирует не только математические навыки, но также навыки чтения и письма. Даже самые маленькие задачи проектирования формулируются в текстовом виде. Без прочных навыков чтения и понимания прочитанного невозможно проектировать программы, которые решают более или менее сложные задачи. И наоборот, методы проектирования программ заставляют разработчика излагать свои мысли правильным и точным языком. Фактически, усваивая рецепт проектирования, учащийся одновременно совершенствует свои навыки артикуляции.

Для иллюстрации взгляните еще раз на описание процесса проектирования в рецепте 1. В нем говорится, что проектировщик должен:

1) проанализировать постановку задачи, обычно обозначаемую словом «задача»;
2) извлечь и абстрактно выразить ее суть;
3) проиллюстрировать суть примерами;
4) определить макет на основе этого анализа;
5) получить результаты и сопоставить их с ожиданиями;
6) доработать продукт с учетом неудачных проверок и тестов.

Каждый шаг требует анализа, описания, точности, сосредоточенности и внимания к деталям. Любой опытный предприниматель, инженер, журналист, юрист, ученый и любой другой профессионал сможет подтвердить, насколько эти навыки важны в повседневной работе. Практика проектирования программ – на бумаге и в DrRacket – это приятный способ приобретения навыков.

Точно так же совершенствование проекта не ограничивается информатикой и созданием программ. Этим занимаются и архитекторы, и композиторы, и писатели, и другие профессионалы. Они начинают с идей в голове и каким-то образом формулируют их суть. Они уточняют идеи на бумаге, пока продукт не будет максимально точно отражать мысленное представление. Воплощая идеи на бумаге, они используют навыки, аналогичные рецептам проектирования: рисование, письмо или игра на музыкальном инструменте, чтобы выразить определенные элементы стиля здания, описать характер че-

ловека или сформулировать элементы мелодии. Их продуктивность в итеративном процессе разработки обусловлена знанием и умением применять базовые рецепты проектирования в своей сфере и правильно выбирать наиболее подходящий для текущей ситуации.

Структура книги

Цель этой книги – познакомить читателей, не имеющих практического опыта, с *системным проектированием программ*. Параллельно она знакомит с *символическим представлением вычислений* – методом объяснения, как работает применение программы к данным. Проще говоря, этот метод обобщает сведения по арифметике и алгебре, которые учащиеся получают в начальной и средней школах соответственно. Но пусть вас это не пугает. В DrRacket имеется механизм пошаговых вычислений, способный иллюстрировать такие вычисления по шагам.

Книга состоит из шести частей, разделенных пятью интермеццо, и обрамляется прологом и эпилогом. Основные части посвящены проектированию программ, а промежуточные интермеццо вводят дополнительные понятия, касающиеся механики программирования и вычислений.

Пролог – это краткое введение в простое программирование. В нем объясняется, как реализовать простую анимацию на `*SL`. Прочитав его, любой новичок почувствует воодушевление и подавленность одновременно. Поэтому в последнем примечании объясняется, почему обычное программирование – это ошибочный путь, и как системный и последовательный подход к проектированию программ устраняет чувство страха, которое обычно испытывает каждый начинающий программист.

За прологом следуют основные части книги.

- **Часть I** описывает наиболее фундаментальные понятия системного проектирования на простых примерах. Основная идея заключается в том, что проектировщики обычно имеют некоторое представление о том, какие данные программа должна принимать и производить. По этой причине при системном подходе к проектированию необходимо извлечь как можно больше подсказок из описания данных, поступающих в программу и исходящих из нее. Для простоты эта часть начинается с атомарных данных – чисел, изображений и т. д., – а затем постепенно вводит новые способы описания данных: интервалы, перечисления, структуры и их комбинации.
- **Интермеццо 1** подробно описывает язык обучения: его словарь, грамматику и значение. Все вместе это обычно называют синтаксисом и семантикой. Разработчики программ используют эту модель вычислений для прогнозирования действий их творения после запуска или для анализа ошибок.

- **Часть II** дополняет часть I средствами описания наиболее интересных и полезных форм данных: составных данных произвольного размера. Программист может продолжать использовать типы данных из части I для представления информации, но эти типы всегда имеют фиксированную глубину и ширину. Эта часть демонстрирует, как небольшое обобщение позволяет перейти к данным произвольного размера. Затем мы переключим свое внимание на системное проектирование программ, обрабатывающих такие данные.
- **Интермеццо 2** вводит краткую и мощную нотацию для записи больших объемов данных: цитирование и антицитирование.
- **Часть III** наглядно демонстрирует сходство многих функций из части II. Никакой язык программирования не должен заставлять программистов создавать фрагменты кода, настолько похожие друг на друга. И наоборот, во всяком хорошем языке программирования есть способы устранения подобного сходства. Ученые-информатики называют этап устранения сходства *абстрагированием*, а его результат – *абстракцией*. Они знают, что абстракции значительно повышают продуктивность программиста. По этой причине в данной части будут представлены рецепты создания и использования абстракций.
- **Интермеццо 3** преследует две цели. Во-первых, здесь вводится понятие *лексической области видимости*, когда язык программирования связывает каждое вхождение имени с его определением, которое программист может найти, просматривая код. Во-вторых, объясняется суть библиотеки с дополнительными механизмами абстракции, включая так называемые *циклы for*.
- **Часть IV** обобщает часть II и явно вводит идею итеративного уточнения в словарь понятий проектирования.
- **Интермеццо 4** объясняет и иллюстрирует, почему десятичные числа работают таким странным образом во всех языках программирования. Представленные здесь основные факты должен знать каждый начинающий программист.
- **Часть V** добавляет новый принцип дизайна. Структурного проектирования и абстракции вполне достаточно для решения большинства задач, с которыми сталкиваются программисты, но иногда этого мало для создания «производительных» программ. То есть программам, созданным с применением принципов структурного проектирования, может потребоваться слишком много времени или энергии для вычисления желаемых ответов. Поэтому ученые-информатики заменяют такие программы, созданные с применением принципов структурного проектирования, программами, способными извлекать выгоду из глубокого понимания предметной области. В этой части книги показано, как спроектировать большой класс именно таких программ.

- **Интермеццо 5** использует примеры из части V для иллюстрации представлений ученых-информатиков о производительности.
- **Часть VI** добавляет последний трюк в инструментарий проектировщиков: аккумуляторы. Если говорить упрощенно, аккумулятор добавляет «память» в функции. Добавление памяти значительно улучшает производительность функций из первых четырех частей книги, созданных с применением принципов структурного проектирования. Для специальных программ из части V аккумуляторы могут даже гарантировать нахождение ответа.
- **Эпилог: что дальше** – это одновременно оценка пройденного и взгляд в будущее.

Читатели, занимающиеся самообразованием самостоятельно, должны проработать всю книгу, от первой до последней страницы. Под словом «проработать» мы подразумеваем, что они должны решить все упражнения или, по крайней мере, знать, как их решать.

Преподаватели тоже должны охватить как можно больше, начиная с пролога и заканчивая эпилогом. Наш опыт преподавания показывает, что это выполнимо. Как правило, мы организуем наши курсы так, чтобы слушатели в течение семестра писали большую и увлекательную программу. Однако мы понимаем, что могут быть обстоятельства, диктующие значительные сокращения, и вкусы некоторых преподавателей требуют иных способов использования книги.

На рис. 1 изображена навигационная схема для тех, кто предпочитает получать знания выборочно. Эта схема представляет собой граф зависимостей. Сплошная стрелка от одного элемента к другому предполагает обязательный порядок чтения; например, прежде чем перейти к части II, обязательно нужно изучить часть I. Пунктирные стрелки, напротив, обозначают предлагаемый маршрут; например, читать пролог перед остальной частью книги необязательно.

Вот три возможных пути изучения книги, которые можно предложить, основываясь на этой схеме:

- преподаватель старшей школы может пройти с учениками (насколько это возможно) части I и II, включая небольшой проект;
- преподаватель университета с квартальной системой обучения может сосредоточиться на части I, части II, части III и части V, а также интермеццо по `*SL` и области видимости;
- преподаватель университета с семестровой системой обучения может предпочесть как можно раньше охватить компромиссы производительности в проектировании. В этом случае мы можем порекомендовать изучить части I и II, а затем раздел об аккумуляторах в части VI, который не зависит от части V. После этого можно углубиться в интермеццо 5 и затем охватить остальную часть книги.

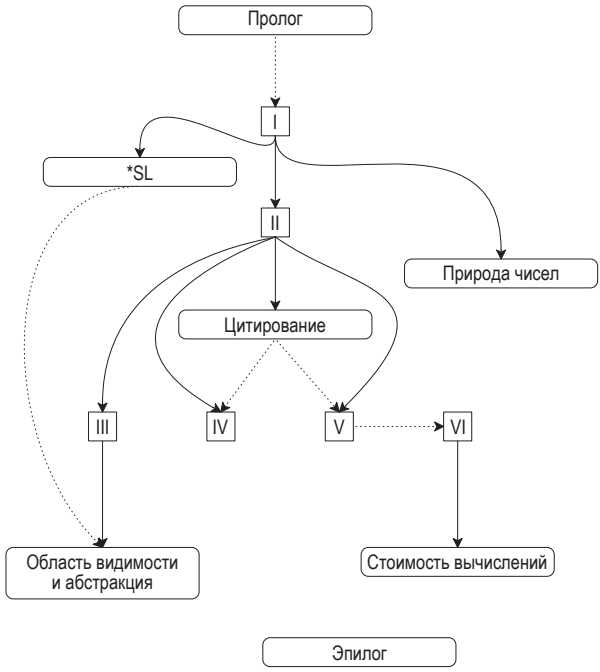

Рис. 1. Зависимости между частями книги и интермеццо

Повторение примеров и упражнений. Повествование в книге снова и снова возвращается к определенным упражнениям и примерам. Например, виртуальные домашние животные встречаются повсюду в части I и иногда даже в части II. Точно так же в обеих частях, I и II, рассматриваются альтернативные подходы к реализации интерактивного текстового редактора. В части V появляются графы, которые перекочевывают в часть VI. Цель этих повторений – последовательное уточнение и закрепление изученного. Мы призываем преподавателей использовать в своей работе эти примеры и упражнения или создать свои подобные последовательности.

Различия между изданиями

Второе издание «Как проектировать программы» имеет несколько важных отличий от первого издания:

1) подчеркивает разницу между проектированием всей программы и отдельных функций, составляющих ее. В частности, в этом издании основное внимание уделяется программам двух типов: управляемым событиями (с графическим интерфейсом и сетевым) и неинтерактивным;
2) проектирование программы на этапе планирования осуществляется сверху вниз, а на следующем за ним этапе построения –

снизу вверх. Мы явно показываем, как интерфейсы библиотек определяют форму элементов программы. В частности, на самом первом этапе проектирования программы создается список желаемых функций. Да, идея списка желаний присутствует и в первом издании, но во втором издании она рассматривается как явный элемент дизайна;

3) выполнимость пункта из списка желаний зависит от рецепта проектирования функции, о чем рассказывается в шести основных частях книги;

4) ключевым элементом структурного проектирования является определение функций, являющихся композицией других функций. Такая композиционная организация особенно полезна в мире неинтерактивных программ. Как и порождающая рекурсия, для правильной композиции требуется *озарение* и признание того факта, что последовательная обработка промежуточных результатов несколькими функциями упрощает общий процесс проектирования. Этот подход тоже требует составить список желаний, но при формулировании этих желаний необходимо тщательно проработать определения промежуточных данных. Это издание книги включает ряд явных упражнений по композиционному проектированию;

Мы благодарим Кэти Фислер (Kathi Fisler) за то, что обратила наше внимание на этот момент.

5) тестирование всегда было частью нашей философии проектирования, однако языки обучения и DrRacket начали обеспечивать достаточно полную его поддержку только в 2002 году, уже после выхода первого издания. Данное новое издание в значительной степени полагается на эту поддержку тестирования;

6) из этого издания мы исключили тему проектирования императивных программ. Старые главы можно найти на нашем сайте[1], а их адаптированные версии войдут во второй том данной серии «How to Design Components»;

7) в этом издании изменены обучающие пакеты с примерами и упражнениями. Предпочтительным стилем связывания этих библиотек считается применение инструкции `require`, но вы все еще можете добавлять их через меню в DrRacket;

8) наконец, во втором издании несколько изменились терминология и обозначения:

Второе издание	**Первое издание**
сигнатура	контракт
детализация	объединение
`'()`	`empty`
`#true`	`true`
`#false`	`false`

Последние три отличия значительно улучшают цитирование списков.

[1] https://htdp.org/2003-09-26/Book/. – *Прим. перев.*

Благодарности из первого издания

Особую благодарность мы хотим выразить четырем нашим коллегам: Роберту Картрайту (Robert «Corky» Cartwright), который совместно с первым автором (Маттиасом Фелляйзеном) разработал вводный курс для университета имени Уильяма Марша Райса; Даниелю П. Фридману (Daniel P. Friedman), предложившему первому автору переписать книгу «The Little LISPer» (также изданную в MIT Press) в 1984 году, которая положила начало этому проекту; Джону Клементсу (John Clements), разработавшему, внедрившему и обслуживающему механизм пошаговых вычислений для DrRacket; и Полу Стеклеру (Paul Steckler), который поддерживал команду, внося свой вклад в разработку нашего набора инструментов программирования.

Участие в создании книги приняли многие другие наши друзья и коллеги, которые использовали ее в своих курсах или давали подробные комментарии к рукописи. Мы благодарны вам за помощь и терпение: Ян Барланд (Ian Barland), Джон Клементс (John Clements), Брюс Дуба (Bruce Duba), Майк Эрнст (Mike Ernst), Кэти Фислер (Kathi Fisler), Даниель П. Фридман (Daniel P. Friedman), Джон Грейнер (John Greiner), Джеральдин Морен (Géraldine Morin), Джон Стоун (John Stone) и Вальдемар Тамез (Valdemar Tamez).

Десятки поколений студентов курса Comp 210 в университете Райса использовали ранние версии текста и предложили различные улучшения. Кроме того, многочисленные участники нашей конференции TeachScheme! использовали рукопись в своих курсах. Многие из них прислали свои комментарии и предложения. Хотелось бы отметить наиболее активных участников: г-жа Барбара Адлер (Ms. Barbara Adler), д-р Стивен Блох (Dr. Stephen Bloch), г-жа Карен Бурас (Ms. Karen Buras), г-н Джек Клей (Mr. Jack Clay), д-р Ричард Клеменс (Dr. Richard Clemens), г-н Кайл Джиллетт (Mr. Kyle Gillette), г-н Марвин Эрнандес (Mr. Marvin Hernandez), г-н Майкл Хант (Mr. Michael Hunt), г-жа Карен Норт (Ms. Karen North), г-н Джейми Рэймонд (Mr. Jamie Raymond) и г-н Роберт Рид (Mr. Robert Reid). Кристофер Фелляйзен (Christopher Felleisen) участвовал в проработке нескольких первых частей книги вместе со своим отцом и помог получить представление о взглядах молодого студента. Хрвое Блажевич (Hrvoje Blazevic, в то время яхтсмен, а ныне капитан танкера *LPG/C Harriette*), Джо Захарий (Joe Zachary, университет штата Юта) и Даниель П. Фридман (Daniel P. Friedman, университет штата Индиана) помогли найти и исправить многочисленные опечатки в первом издании. Спасибо всем вам.

Наконец, Матиас выражает свою благодарность Хельге (Helga) за ее многолетнее терпение и за окружение уютом ее рассеянных мужа и отца. Робби выражает благодарность Синь-Хуэй Хуан (Hsing-Huei Huang) – без ее поддержки и терпения он не смог бы работать так плодотворно. Мэтью благодарит Вэнь Юань (Wen Yuan) за ее постоянную поддержку и непреходящую музыку. Шрирам выражает призна-