1 Почему я написал эту книгу

Когда я был студентом-физиком, меня завораживала мысль, что мир можно описать простыми уравнениями, такими как $F = ma$. Позже, став программистом, а затем ученым в области компьютерных наук, я увлекся формальными методами, ведь они, казалось, могли дать нечто подобное для программного обеспечения: кратко выразить сущность ПО с помощью логики.

Страсть к проектированию

За 30 лет, прошедших с момента получения докторской степени, моим главным вкладом как ученого стал Alloy[3] — язык для описания и автоматического анализа программных решений. Эта работа оказалась захватывающей и принесла мне массу удовольствия, но со временем я пришел к выводу, что суть ПО заключается вовсе не в логике или анализе. Меня увлек совсем не тот вопрос, который занимал умы большинства исследователей формальных методов — соответствие поведения программы ее спецификации, — а вопросы *проектирования*[4].

Под проектированием я здесь имею в виду то же, что подразумевают в других дисциплинах: придание формы какой-либо вещи для удовлетворения запросов человека. Проектирование, как выразился архитектор Кристофер Александер, заключается в создании *формы*, соответствующей *контексту*. Для ПО это означает определение его поведения: какие элементы управления оно предложит и как будет реагировать на действия пользователей. На эти вопросы нет правильных или неправильных ответов, есть только более или менее подходящие варианты[5].

Меня интересовало, почему одни программные продукты выглядят естественно и элегантно, предсказуемо реагируют (как только вы изучаете их основы) и их можно эффективно сопрягать друг с другом, а другие кажутся перегруженными и ведут себя непредсказуемо и непоследовательно. Должна же существовать какая-то ключевая концепция или теория проектирования программного обеспечения, объясняющая, почему одни программные продукты удачны, а другие — нет, и помогающая устранять проблемы, а также предупреждать их.

Проектирование в области компьютерных наук и других областях

В области моих интересов (формальные методы, программная инженерия и языки программирования) я нашел теорию, которая определяет так называемый внутренний проект, буквально — структуру кода. У программистов есть развитый язык для описания проектирования и устоявшиеся критерии того, что отличает хорошие решения от плохих. Но подобного языка или критериев для проектирования ПО, ориентированного на пользователя, не существует. Нет проекта, который определяет восприятие программного обеспечения как некой сущности в конкретном контексте[6].

Внутренний проект очень важен для кода и в первую очередь влияет на то, что инженеры-программисты называют легкостью сопровождения, — насколько легко (или сложно) изменять код с течением времени по мере его развития. Это также влияет на производительность и надежность. Однако ключевые решения, определяющие, являются ли приложение и система полезными и удовлетворяют ли потребностям своих пользователей, находятся в другой области, а именно, в области проектирования ПО, в которой формируются функциональность и модели взаимодействия с пользователем.

Эти ключевые вопросы когда-то занимали центральное место в информатике. В области программной инженерии их обсуждали на семинарах по разработке, создавали спецификации и требования. В сфере взаимодействия человека и компьютера эти вопросы встречались в работах по графическим пользовательским интерфейсам (GUI) и вычислительным моделям поведения пользователей, которые публиковались на заре развития компьютерных наук[7].

Но со временем такие вопросы стали менее популярными и их перестали поднимать. Исследования в области программной инженерии стали менее широкими. Устранение дефектов — будь то с помощью тестирования или

более сложных средств, таких как верификация программ, — стало синонимом качества ПО[8]. Но этим трудно чего-то добиться: если ваше программное обеспечение неправильно спроектировано, никакие меры по устранению дефектов не помогут и придется возвращаться к самому началу и править сам проект[9].

В исследованиях взаимодействия человека и компьютера (human-computer interaction, HCI) акцент сместился к новым технологиям взаимодействия, инструментам и фреймворкам, нишевым областям и смежным дисциплинам (таким как этнография и социология). И программная инженерия, и HCI с энтузиазмом восприняли переход на эмпирические методы в тщетной надежде повысить свою значимость. Но вместо этого спрос на конкретные показатели и успешность, по-видимому, привел исследователей к менее амбициозным проектам, для которых допустима более легкая оценка, и затормозил прогресс в решении более масштабных и значимых вопросов[10].

Удивительно, но, даже несмотря на то что интерес к проектированию заметно ослаб, разговоры о нем ведутся повсюду. И здесь нет никакого противоречия: говорят почти всегда о самом *процессе* проектирования, будь то дизайн-мышление (привлекательная методика итерационного подхода) или гибкая разработка ПО. Эти процессы, несомненно, важны (при условии разумного применения, а не как панацея), но по большей части они бессодержательны. Я ни в коем случае не пытаюсь принизить их значимость, а просто описываю их сущность. Дизайн-мышление, например, способно помочь вам в разработке решения в соответствии с вашим пониманием проблемы или чередовать фазы мозгового штурма (дивергенция) и отсева идей (конвергенция). Но ни в одной книге по дизайн-мышлению, что мне встречались, не рассказывается в подробностях о каких-либо конкретных проектах и о том, как дизайн-мышление облегчает проектирование. Возможно, именно независимость дизайн-мышления от предмета проектирования делает его столь привлекательным, а также объясняет, почему оно мало что может сказать о более глубоких проблемах проектирования в конкретной области, такой как создание программного обеспечения[11].

Ясность и простота дизайна

Когда я начинал проект Alloy, целью которого было создание языка проектирования, подходящего для автоматизированного анализа, я критически относился к существующим языкам моделирования и спецификации. Отсутствие в них инструментальной поддержки фактически делало их недоступными для понимания. И эта нелестная характеристика не была безосновательной.

В конце концов, зачем утруждать себя созданием сложной проектной модели, если потом с ней ничего нельзя сделать? В частности, я утверждал, что усилия проектировщика должны немедленно вознаграждаться применением автоматизации, которая мгновенно дает обратную связь в виде неожиданных сценариев, предоставляющих больше пищи для размышления[12].

Я думаю, что был прав: автоматизация в Alloy действительно изменила подход к проектному моделированию. Но я недооценил важность документирования проекта. На самом деле исследователи формальных методов, стремящиеся продемонстрировать эффективность своих инструментов, знали и скрывали (хоть и не слишком усердно) то, что многие ошибки выявлялись еще *до* применения инструментов! Простого преобразования проекта в логику было достаточно для обнаружения серьезных проблем. Исследователь в области программного обеспечения Майкл Джексон объяснял это не логикой как таковой, а именно сложностью ее использования. Однажды он с юмором предположил, что качество программных систем можно повысить, если просто обязать проектировщиков вести свои проекты на китайском языке.

Четкость хороша не только для выявления недостатков проекта постфактум. Это в первую очередь основное требование к хорошему проекту. Преподавая программирование и программную инженерию на протяжении последних 30 лет, я все больше убеждаюсь в том, что определяющим фактором успеха при разработке ПО является не использование новейших языков программирования и инструментов, не методология, которой вы следуете (например, agile), и даже не структура кода. Главное — иметь четкое представление о том, чего вы пытаетесь добиться. Если ваши цели ясны, а план понятен и видно, как он помогает достичь этих целей, то ваш код, как правило, тоже окажется понятным. И если что-то пойдет не так, будет легко определить, как это исправить[13].

Именно эта ясность отличает выдающееся программное обеспечение от остального. Когда в 1984 году появился Apple Macintosh, люди сразу разобрались, как упорядочить файлы с помощью папок. Сложности предыдущих операционных систем (таких как Unix, где даже простое перемещение файлов требовало усилий), казалось, остались в прошлом.

Но что именно представляет собой эта ясность и как ее добиться? Еще в 1960-х годах стало понятно, что концептуальные модели являются основой основ. Задача состояла не просто в том, чтобы *донести* до пользователя концептуальную модель ПО, совпадающую с замыслом программистов, но и рассматривать ее как самостоятельный объект проектирования. С правильно

построенной концептуальной моделью программное обеспечение получалось понятным и, следовательно, удобным в использовании. Это была отличная идея, но, похоже, никто ее так и не реализовал и «концепции» так и остались расплывчатым, хотя и вдохновляющим понятием[14].

Как появилась эта книга

Я был убежден в том, что концептуальные модели на самом деле являются сутью программного обеспечения, и около восьми лет назад начал выяснять, как их можно было бы определить. Мне хотелось дать им конкретное содержание, чтобы указать на концептуальную модель некоего ПО, сравнить ее с другими (и с представлением пользователей о ПО) и иметь четкий фокус во время обсуждения проекта.

Эта задача не казалась такой уж трудной: в качестве первого шага можно было бы описать поведение ПО, абстрагировавшись от деталей, дабы исключить случайные и «неконцептуальные» аспекты (например, особенностей интерфейса). Но сложность заключалась в нахождении подходящей структуры для модели. У меня было подозрение, что концептуальная модель должна состоять из концепций, но я не знал, что это такое.

Возьмем, к примеру, такую социальную сеть, как Facebook. Там должна быть концепция, связанная с лайками. Конечно, это не только функция или действие (например, нажатие кнопки «Нравится»), поскольку подобных элементов слишком много и они отражают лишь часть идеи. Концепция лайка, безусловно, не просто объект или сущность, которые создаются при нажатии кнопки. Как минимум она отражает отношение между вещами и их подобиями. Мне также показалось важным, чтобы понятие «лайк» не ассоциировалось с какими-либо конкретными вещами: вы могли отметить посты, комментарии, страницы и т. д. На языке программирования такая концепция называется универсальной или полиморфной.

Начало обсуждения

Данная книга — результат моих исследований на сегодняшний день. Рассмотрев десятки проблем с проектированием, возникающих в популярных приложениях, я придумал новый подход к разработке ПО. Приятным дополнением этого проекта оказалось то, что в каждом сбое приложения или ошибке в работе была и положительная сторона: возможность расширить мою коллекцию примеров. А анализ позволил мне глубже понять, с какими сложностями сталкиваются проектировщики, и еще больше уважать их труд.

Конечно, проблема проектирования ПО далека от решения. Но, как однажды мудро подметила Кирстен Олсон, моя подруга, цель книги — начать обсуждение, а не завершить его. В ходе многочисленных выступлений, посвященных теме книги, я с удивлением обнаружил, что эта тема находит у аудитории гораздо больший отклик, чем любая другая из ранее мной поднятых. Я подозреваю, что это происходит потому, что проектирование ПО — это то, что мы все хотим обсудить, но не знаем, с чего начать.

Итак, мои читатели — коллеги-исследователи, проектировщики и пользователи, — с помощью этой книги я приглашаю вас к началу плодотворного и приятного обсуждения, которое, я надеюсь, будет продолжено.