Думается, что важным дополнением к методам объектно-ориентированного проектирования как раз и являются наши паттерны. Они показывают, как применять такие базовые приемы, как объекты, наследование и полиморфизм, демонстрируют, как можно параметризовать систему алгоритмом, поведением, состоянием или видом объектов, которые предполагается создавать. Паттерны проектирования позволяют не просто зафиксировать результаты решений, а ответить на многочисленные «почему», возникающие в ходе проектирования. Разделы «Применимость», «Результаты» и «Реализация» в описаниях паттернов помогут вам сориентироваться при принятии решения.

Паттерны проектирования особенно полезны тогда, когда нужно преобразовать аналитическую модель в модель реализации. Вопреки многочисленным заверениям о беспрепятственном переходе от объектно-ориентированного анализа к проектированию, на практике этот процесс никогда не происходит гладко. В гибком проекте, ориентированном на повторное использование, имеются объекты, которых нет в аналитической модели. На проект оказывают влияние выбранные язык программирования и библиотеки классов. Аналитические модели часто приходится пересматривать, чтобы обеспечить повторное использование. Многие паттерны, включенные в каталог, непосредственно связаны с такого рода вопросами, почему мы и называем их паттернами проектирования.

Полноценная методика проектирования основывается не только на паттернах проектирования. Тут могут быть паттерны и анализа, и пользовательского интерфейса, и оптимизации производительности. Но паттерны – очень важная составная часть методики проектирования, которая до сих пор отсутствовала.

Цель реорганизации

Повторно используемое программное обеспечение часто приходится реорганизовывать [OJ90]. Паттерны проектирования помогают вам решить, как именно реорганизовать проект, и могут уменьшить вероятность реорганизации в будущем.

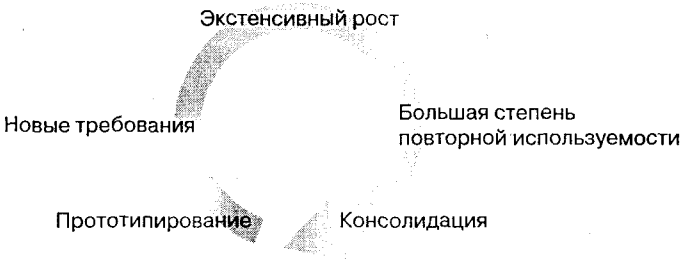
По Брайану Футу (Brian Foote), жизненный цикл объектно-ориентированной программы состоит из трех фаз: *прототипирования*, *экстенсивного роста* и *консолидации* [Foo92].

Фаза прототипирования характеризуется высокой активностью проектировщиков, направленной на то, чтобы программа начала работать. При этом быстро создается прототип, который последовательно изменяется до тех пор, пока не будут решены первоначальные задачи. Обычно на данном этапе программа представляет собой набор иерархий классов, отражающих сущности предметной области. Основной вид повторного использования – это прозрачный ящик и наследование.

После ввода в эксплуатацию развитие программы определяется двумя противоречивыми факторами: программе необходимо отвечать все новым требованиям и одновременно должна повышаться степень ее повторной используемости. Новые требования диктуют необходимость добавления новых классов и операций, быть может, даже целых иерархий классов. Эти требования могут удовлетворяться, когда начинается фаза экстенсивного роста программы. Но данный период не может продолжаться долго. В конце концов, программа становится слишком негибкой и не поддается дальнейшим изменениям. Иерархии классов более не соответствуют

одной предметной области. Напротив, они отражают множество разных предметных областей, а классы определяют совершенно разнородные операции и переменные экземпляров.

Чтобы программа могла развиваться и дальше, ее необходимо пересмотреть и реорганизовать. Именно в этот период часто создаются каркасы. При реорганизации классы, описывающие специализированные и универсальные компоненты, отделяются друг от друга, операции перемещаются вверх и вниз по иерархии, а интерфейсы классов становятся более рациональными. В фазе консолидации появляется много новых объектов, часто в результате декомпозиции существующих и использования композиции вместо наследования. Поэтому прозрачный ящик заменяется черным. В связи с непрекращающимся потоком новых требований и стремлением ко все более высокой степени повторной используемости объектно-ориентированная программа должна вновь и вновь проходить через фазы экстенсивного роста и консолидации.



От этого цикла никуда не деться. Но хороший проектировщик предвидит изменения, которые могут потребовать реорганизации. Он знает, какие структуры классов и объектов позволят избежать реорганизации, его проект устойчив к изменениям требований. Требования проанализированы, выявлены те из них, которые, по всей вероятности, изменятся на протяжении жизненного цикла программы, и в проекте учтено все это.

В наших паттернах проектирования нашли свое применение многие структуры, появившиеся в результате реорганизации. Использование паттернов на ранних стадиях проекта предотвратит реорганизацию в будущем. Но даже если вы не увидели, как применить паттерн, пока не создали всю систему, он все равно «подскажет», как ее можно изменить.

6.2. Краткая история

Начало этому каталогу положила докторская диссертация Эриха [Gam91, Gam92]. Почти половина всех паттернов была представлена в этой работе. К моменту проведения конференции OOPSLA '91 диссертацию официально признали независимым каталогом, и для продолжения работы над ним к Эриху присоединился Ричард. Вскоре после этого к компании примкнул и Джон. Незадолго до конференции OOPSLA '92 в группу влился Ральф. Мы изо всех сил старались, чтобы каталог был готов к публикации в трудах ECOOP '93, но вскоре осознали,