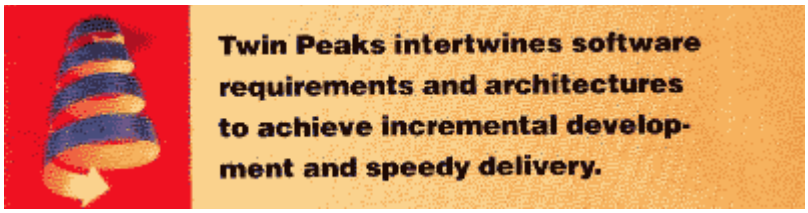
Соединение требований и архитектур

Башар Нузейбе (**Bashar Nuseibeh**), Открытый университет

Непреодолимые экономические аргументы объясняют, почему раннее понимание требований акционеров приводит к системам, удовлетворяющим их ожидания. Одинаково непреодолимые аргументы оправдывают раннее понимание и истолкование архитектуры программных систем для обеспечения основы для раскрытия дальнейших требований и ограничений, оценки технической осуществимости системы и определения альтернативных конструкторских решений.

Организации-разработчики программного обеспечения часто выбирают между двумя альтернативными начальными точками – требованиями и архитектурами. Это неизменно ведёт к водопадному процессу разработки, который производит искусственно замороженные документы по требованиям для использования на следующем шаге жизненного цикла разработки. Точно так же этот процесс создаёт системы с ограниченными архитектурами, которые ограничивают пользователей и мешают разработчикам сопротивлением неизбежным и желаемым изменениям в требованиях.

Спиральная модель жизненного цикла управляется со многими недостатками водопадной модели обеспечением пошагового процесса разработки, в котором разработчики неоднократно оценивают меняющиеся риски проекта для управления неустойчивыми требованиями и запасами. Одинаково мелкозернистая спираль жизненного цикла отражает как реалии, так и настоятельные потребности современного процесса разработки программного обеспечения. Такой жизненный цикл подтверждает необходимость разработки программных архитектур, которые стабильны и при этом легко приспосабливаемы в присутствии меняющихся требований. Краеугольный камень этого процесса – это то, что разработчики «мастерят» требования к системе и её архитектуру одновременно и чередуют свой процесс разработки (В. Свартаут и Р. Балзер. Неизбежное переплетение спецификации и реализации / *Комм. АКМ*, выпуск 25, № 7. – 1982. – С. 438-440 [W. Swartout and R. Balzer, “On the Inevitable Intertwining of Specification and Implementation”, *Comm. ACM*, vol. 25, no. 7, 1982, pp. 438-440]).



МОДЕЛЬ ДВОЙНЫХ ВЕРШИН

За исключением хорошо определённых проблемных областей и чётко оговоренных процедур, большинство проектов разработки программного обеспечения одновременно обращаются к спецификации требований и к результатам проектирования – и это оправдано. Достижение разделения требований и шагов создания часто является трудным, потому что их искусственное упорядочение заставляет разработчиков сосредотачивать внимание на обоих аспектах в любой момент времени. В действительности же, возможные архитектуры могут удерживать проектировщиков от соответствия отдельным требованиям, а выбор требований может влиять на архитектуру, которую проектировщик выбирает или разрабатывает.

Основываясь на нашем опыте в проектах разработки промышленного программного обеспечения, я и мои коллеги используем адаптацию спиральной модели жизненного цикла. Мы неформально называем её «моделью двойных вершин», чтобы подчеркнуть равное положение, которое мы отводим требованиям и архитектурам. Хотя эта модель

развивает требования и архитектурные спецификации одновременно, она продолжает отделять структуру и спецификацию проблемы от структуры и спецификации решения в повторяющемся процессе, что приводит к прогрессивно более подробным требованиям и проектным спецификациям, как показано на рисунке 1.

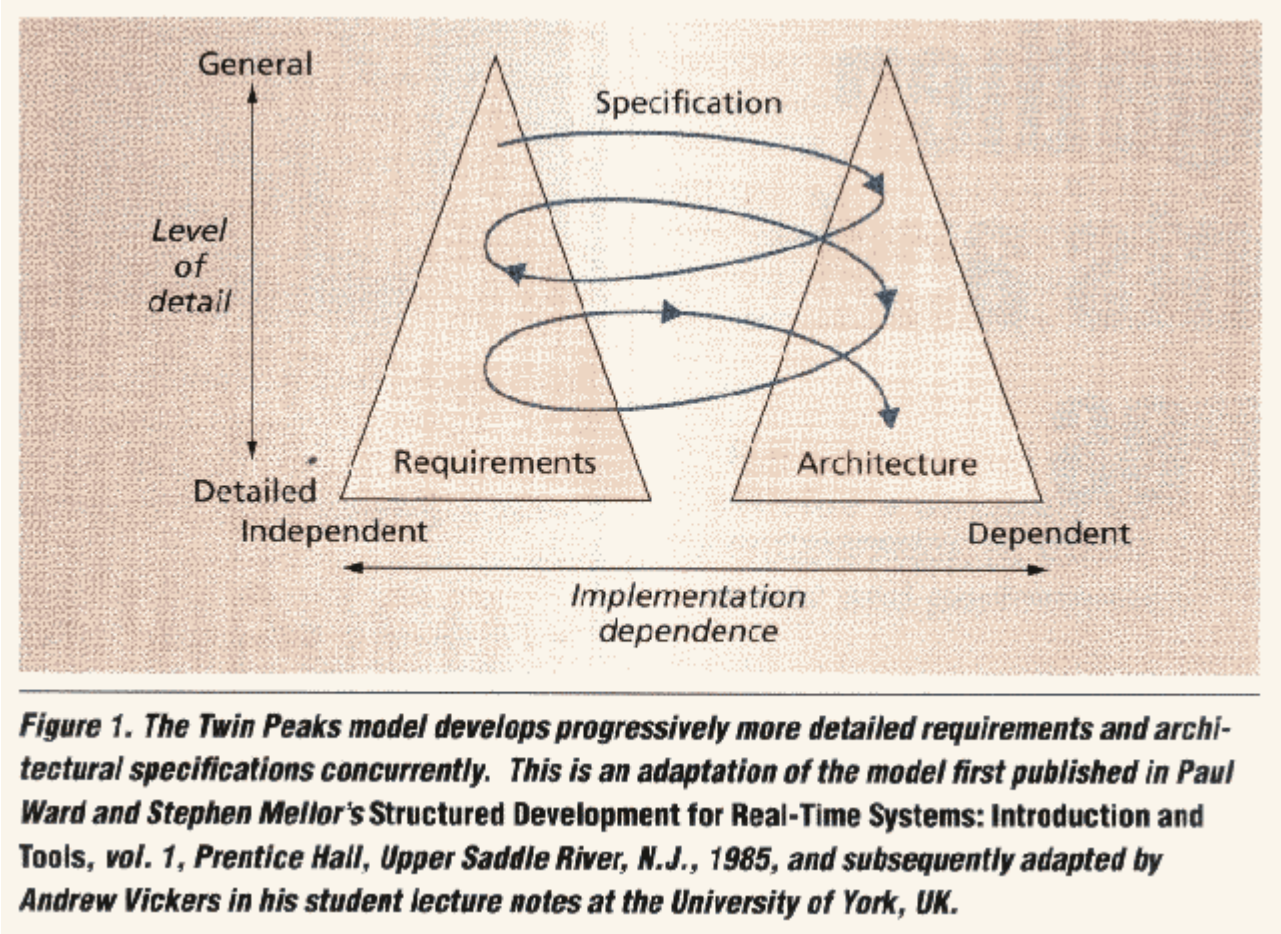


Figure 1. The Twin Peaks model develops progressively more detailed requirements and architectural specifications concurrently. This is an adaptation of the model first published in Paul Ward and Stephen Mellor's* Structured Development for Real-Time Systems: Introduction and Tools, *vol. 1, Prentice Hall, Upper Saddle River, N.J., 1985, and subsequently adapted by Andrew Vickers in his student lecture notes at the University of York, UK.

Здесь: *Requirements* – требования, *Architecture* – архитектура,
Specification – спецификация;
Level of detail – уровень детализации (общий – подробный);
Implementation dependence – зависимость реализации (растёт слева направо).

Модель двойных вершин основывается на трёх принципах менеджмента, сформулированных Барри Боэмом (Barry Boehm) (Требования, которые используют принципы ЯУЭКЯУЭ, КГПО и «быстрое изменение» / *Компьютер*, № 6. – 2000. – С. 99-102 [“Requirements that Handle IKIWISI, COTS, and Rapid Change”, Computer, July 2000, pp. 99-102]):

- *Я Узнаю Это, Когда Я Увижу Это (ЯУЭКЯУЭ) (IKIWISI, I'll Know It When I See It).* Часто требования появляются только после того, как пользователи получили возможность посмотреть и «потрогать» модели или прототипы. Модель двойных вершин явно разрешает пользователю исследовать предлагаемые решения на ранних этапах, позволяя использовать пошаговую разработку и последовательное управление риском.
- *Коммерческое Готовое Программное Обеспечение (КГПО) (COTS, Commercial Off-The-shelf Software).* Всё больше и больше разработка программного обеспечения в действительности представляет собой процесс распознавания и выбора желаемых требований из существующих коммерчески доступных пакетов программного обеспечения. С моделью двойных вершин разработчики могут распознавать требования и подбирать архитектуры из коммерчески доступных продуктов, быстро и

постепенно. Разработчик получает выгоду путём быстрого сужения выборки или принятия ключевых архитектурных решений для согласования с существующими готовыми решениями.

- *Быстрое изменение (Rapid Change).* Реакция на изменения продолжает быть основной проблемой при разработке программного обеспечения и в управлении проектами. Сосредотачиваясь на мелкозернистой разработке, модель двойных вершин восприимчива к изменениям по мере их появления. Анализ и распознавание требований ядра программной системы – то, что необходимо для разработки стабильной архитектуры программной системы в среде меняющихся требований.

Разработка программных систем в этом контексте требует рассмотрения других процессов разработки. Обращение к принципу ЯУЭКЯУЭ означает начало проектирования и реализации раньше, чем обычно; использование принципа КГПО требует рассмотрения возможности повторного использования на более ранних этапах спецификации требований; сохранение конкурентоспособности при приспособлении к быстрым изменениям требует от нас выполнять все задачи разработки ещё быстрее.

ПОШАГОВОЕ ПОСТРОЕНИЕ МОДУЛЬНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Построение системы с хорошо определёнными интерфейсами компонентов предоставляет возможности эффективного повторного использования и эксплуатации. Непонятно, тем не менее, как компонентно-ориентированные подходы к разработке вписываются в сам процесс разработки. Один подход нужен для рассмотрения использования требований, архитектуры и проектных *шаблонов*. Люди, занимающиеся проектированием программного обеспечения, уже выделили *шаблоны проектов* для выражения спектра реализаций. Люди, занимающиеся архитектурами программного обеспечения, выделили подходящие архитектурные стили для соответствия различным глобальным требованиям. Люди, разрабатывающие требования, посодействовали распространению использования *проблемных описаний (problem frames)* Майкла Джексона (Michael Jackson) и *шаблонов анализа (analysis patterns)* Мартина Фоулера (Martin Fowler) для распознавания проблем, решения которых существуют.

Какие связи соединяют эти различные шаблоны? На рисунке 2 показано, что мы можем рассматривать шаблоны требований, проектов и архитектур как отправную точку для разработки, основанной на компонентах. Например, некая данная неизменная архитектура может ограничивать виды проблем, которые мы можем рассматривать, и возможные проектные решения, которые мы может разработать, тогда как жёсткие требования могут ограничивать возможные архитектуры и проектные решения.

Модель двойных вершин представляет собой многое из существующей, но не выраженной явно практики разработки программного обеспечения

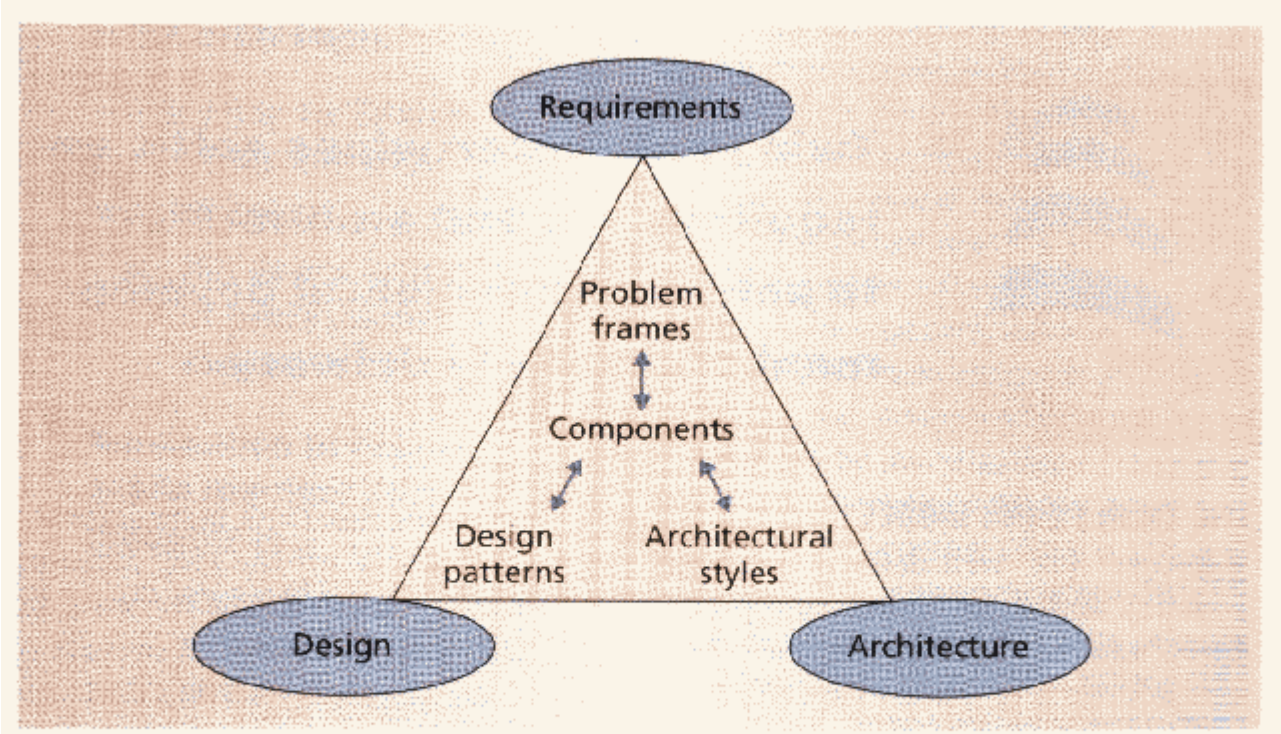


Figure 2. Part of the software-development terrain, with requirements, architecture, and design receiving similar attention. Patterns of each affect the kind of system (components) developed, and the relationship between them is a key determinant of the kind of process developers adopt.

Здесь: *Requirements* – требования, *Architecture* – архитектура,
Design – проект (проектное решение);
Components – компоненты;
Problem frames – проблемные описания,
Architectural styles – архитектурные стили,
Design patterns – шаблоны проектов (проектных решений).

С точки зрения технологии разработки требований, достижение удовлетворительного структурирования проблемы путём как можно более раннего использования проблемных описаний является необходимым. Учитывая то, что существующие архитектуры могут влиять на то, как разработчики структурируют проблемы, некоторые проблемные описания могут нуждаться в реинжиниринге существующих архитектурных решений.

ЕДИНЕНИЕ ПРОЦЕССА РАЗРАБОТКИ

У модели двойных вершин есть много общего с экстремальным программированием (ЭП) Кента Бека (Kent Beck), как-то цель исследования возможностей реализации с самого начала и последовательно далее. Модель двойных вершин дополняет ЭП тем, что сосредотачивается на «краеугольных камнях» разработки программного обеспечения – требованиях и архитектурах. Это потенциально позволяет управляться с вопросами масштабирования, которые часто считаются слабой стороной ЭП.

Раннее понимание требований и выбор архитектуры – это ключ к управлению большими системами и проектами. ЭП сосредотачивается на получении кода, иногда в ущерб более широкой картине требований и архитектур.

Конечно, сосредоточение на требованиях и архитектурах как таковых недостаточно для достижения масштабируемости. Модульность и итерации также являются

решающими. Модель двойных вершин по природе своей итеративна, а соединение её с опробованными и тестированными компонентами, полученными из хорошо понимаемых шаблонов, может облегчить пошаговую разработку больших систем. В результате, весь процесс разработки программного обеспечения неизбежно идёт более сложным путём от проблемы к решению.

Хотя понятийные различия между требованиями и проектом теперь гораздо лучше понимаются и разделяются, сам *процесс* продвижения между проблемным миром и миром решений не так хорошо распознаваем (Майкл Годик и Башар Нузейбе. Дорога процесса между требованиями и проектом / 2я Всемирная конференция по интегрированному проектированию и технологии процессов. – Остин Техас, 1996. – С. 176-177 [Michel Goedick and Bashar Nuseibeh, “The Process Road between Requirements and Design”, *Proc. 2nd World Conf. Integrated Design and Process Technology*, SDPS, Austin Texas, 1996, pp. 176-177]). Исследователи и практики борются за разработку процессов, которые позволяют осуществлять быструю разработку на конкурирующем рынке, соединённую с улучшенным анализом и планированием, что необходимо для производства высококачественных систем в сжатые сроки и при ограничениях бюджета.

Более здравый и реалистичный процесс разработки позволяет как инженерам по требованиям, так и системным архитекторам работать одновременно и итеративно для описания тех артефактов, которые они желают произвести. Этот процесс позволяет разработчикам лучше понимать проблемы через рассмотрение архитектурных ограничений, и они могут разрабатывать и приспосабливать архитектуры, основываясь на требованиях.

Многие трудные вопросы остаются без ответа:

- Какие программные архитектуры (или архитектурные стили) стабильны в присутствии меняющихся требований, и как мы выбираем их?
- Какие классы требований более стабильны, чем другие, и как мы выделяем их?
- Какие типы изменений могут переживать системы на протяжении своей жизни, и как мы управляем требованиями и архитектурами (и процессом их разработки), чтобы минимизировать последствия этих изменений?

Ответы на эти вопросы будут влиять на основные возникающие варианты контекста разработки программного обеспечения, включая:

- *линейки и семейства продуктов*, которые нуждаются в стабильных архитектурах, выдерживающих изменения требований;
- *готовые системы*, которые требуют распознавания и подбора существующих архитектур в соответствие требованиям (в противопоставление разработке систем с нуля);
- *наследственные системы*, которые могут объединять существующие системные ограничения в спецификации требований.

Процессы, которые основываются на характеристиках модели двойных вершин, – это первые шаги на пути закрепления необходимости архитектурной стабильности при неизбежном непостоянстве требований.

Процессы разработки, которые способствуют быстрому, пошаговому вводу во владение, необходимы для программных систем, которые нуждаются в быстрой разработке, с прогрессивно сокращённым временем представления конечного продукта на рынке в качестве ключевого требования. Модель двойных вершин представляет собой многое из существующей, но не выраженной явно практики разработки программного обеспечения. Несмотря на то, что она основывается на принятых исследованиях её

эволюционного развития, разработчики программного обеспечения ещё не признали, что такая модель отражает принятую практику. ☼

Башар Нузейбе *(**Bashar Nuseibeh**) является профессором информатики и вычислительной техники в Открытом университете, что в Великобритании, и директором Центра разработки системных требований при кафедре информатики и вычислительной техники Британского колледжа в Лондоне. С ним можно связаться по адресу B.A.Nuseibeh@open.ac.uk или посетить его страничку в Интернете – http://mcs.open.ac.uk/ban25.*

Редактор: Барри Боэм (Barry Boehm), кафедра информатики и вычислительной техники Университета Южной Калифорнии, Лос-Анджелес, CA 90089; boehm@sunset.usc.edu.