Глава 2
Процесс Harmony

В этой главе:

- Процесс разработки Harmony
  - Зачем нужен процесс?
  - Обзор процесса Harmony
    - Основные возможности
    - Временные рамки процесса
  - Разновидности процесса Harmony
  - Микроцикл Harmony
    - Вечеринка!
    - Анализ
    - Проектирование
    - Трансляция
    - Тестирование

2.1 Введение

Составляющими методологии являются язык, используемый для спецификации интересующих нас элементов и отношений, и процесс, который указывает разработчикам, какие элементы языка, как и когда необходимо использовать. В процессе Harmony[1] используется язык UML и его разновидности в виде профилей UML, такие как язык моделирования систем SysML или архитектурный каркас министерства обороны США DoDaf. Процесс Harmony также определяет интегрированный набор последовательностей работ, помогающих разработчикам использовать все преимущества языка UML для разработки стабильных, функциональных и безопасных систем.

Число различных процессов разработки, используемых в настоящее время, весьма велико: от "да зачем вообще нам нужен этот чертов процесс" до очень строгих, формализованных процессов. В начале данной главы делается обзор двух основных разновидностей процесса Harmony, после чего определяются детальные последовательности работ для каждой из фаз процесса.

[1] Процесс Harmony по сути является следующей версией процесса ROPES (Rapid Object-oriented Process for Embedded Systems), обсуждавщегося автором в предыдущих книгах, и расширенный для целей разработки систем.

Начиная со следующей главы, вам будут предлагаться задачи над которыми вы будете работать. Задачи будут предлагаться приблизительно в том порядке, в котором они возникают если вы работает в соответствии с процессом Harmony. Предлагаемый порядок следования задач может отличаться от последовательности их возникновения, если бы вы использовали какой-либо другой процесс.

2.2 Процесс разработки Harmony

Процесс определяет интегрированный набор последовательностей работ. Каждый из таких последовательностей работ представляет отдельный аспект, как правило – фазу процесса, и содержит подробную информацию о том, какие действия необходимо выполнять всем участникам процесса, а также когда и как их следует выполнять, и какие артефакты создаются при этом.

Хороший процесс представляет собой руководство по эффективному способу разработки высоконадежных систем с минимальными затратами. Если последовательностей работ слишком много, это означает что процесс либо отсутстствует, либо время высококвалифицированных разработчиков и ресурсы проекта нерационально расходуются на пустую работу, например на создание множества бесполезных документов. В хорошем процессе, как правило, создаются несколько "бумажных" артефактов – но только таких, которые имеют реальную ценность; но даже для них во главу угла ставятся соображения рентабельности.

2.2.1 Зачем нужен процесс?

Основная причина по которой нам, разработчикам программного обеспечения и систем, стоит задуматься об использовании хорошего процесса заключаются в том, что он позволяют облегчить нашу жизнь и улучшить наши продукты. В частности, хороший процесс:

- Предоставляет шаблон проекта, которым разработчики могут руководствоваться в процессе разработки и поставки продукта;
- Улучшает качество продукта путем:
  - Уменьшения количества дефектов;
  - Уменьшения серьезности дефектов;
  - Облегчения повторного использование;
  - Улучшения стабильности и возможности сопровождения;
- Повышения предсказуемости проекта:
  - общего объема работ;
  - времени, необходимого для их выполнения;
- Позволяет донести необходимую информацию по проекту до различных заинтересованных лиц в виде, позволяющем им воспользоваться ею наиболее эффективно;

Если используемый вами процесс не позволяет достигнуть этих целей, то скорее всего это *плохой* процесс и вам необходимо всерьез задуматься о его замене на лучший. Данные цели *могут быть* достигнуты при помощи хорошего процесса, в то время как плохой процесс *помешает* вам их достигнуть.

Итак, что такое процесс? Для Harmony мы даем следующее определение процесса:

> ***Процесс*** *представляет собой спецификацию упорядоченного набора деятельностей, выполняемых участниками проекта при совместной работе, результатом которой является согласованный набор артефактов проекта, одним из которых является результирующая система.*

Процесс определяет *роли участников* -- те "костюмы", которые "носят" *участники* процесса, выполняющие ту или иную деятельность в проекте. Результатом каждого вида деятельности является создание или изменение одного или нескольких *артефактов*. Например, большинство процессов содержат деятельность по сбору требований, которая выполняется до начала проектирования. Эта деятельность выполняется аналитиком требований (участником процесса), действующим как разработчик модели ПО (роль участника), в результате которой может создаваться артефакт - модель программного обеспечения, на основе которой будет сгенерирован программный код. На Рис. 2-1 отображены эти фундаментальные аспекты и отношения, присущие процессу по разработке.

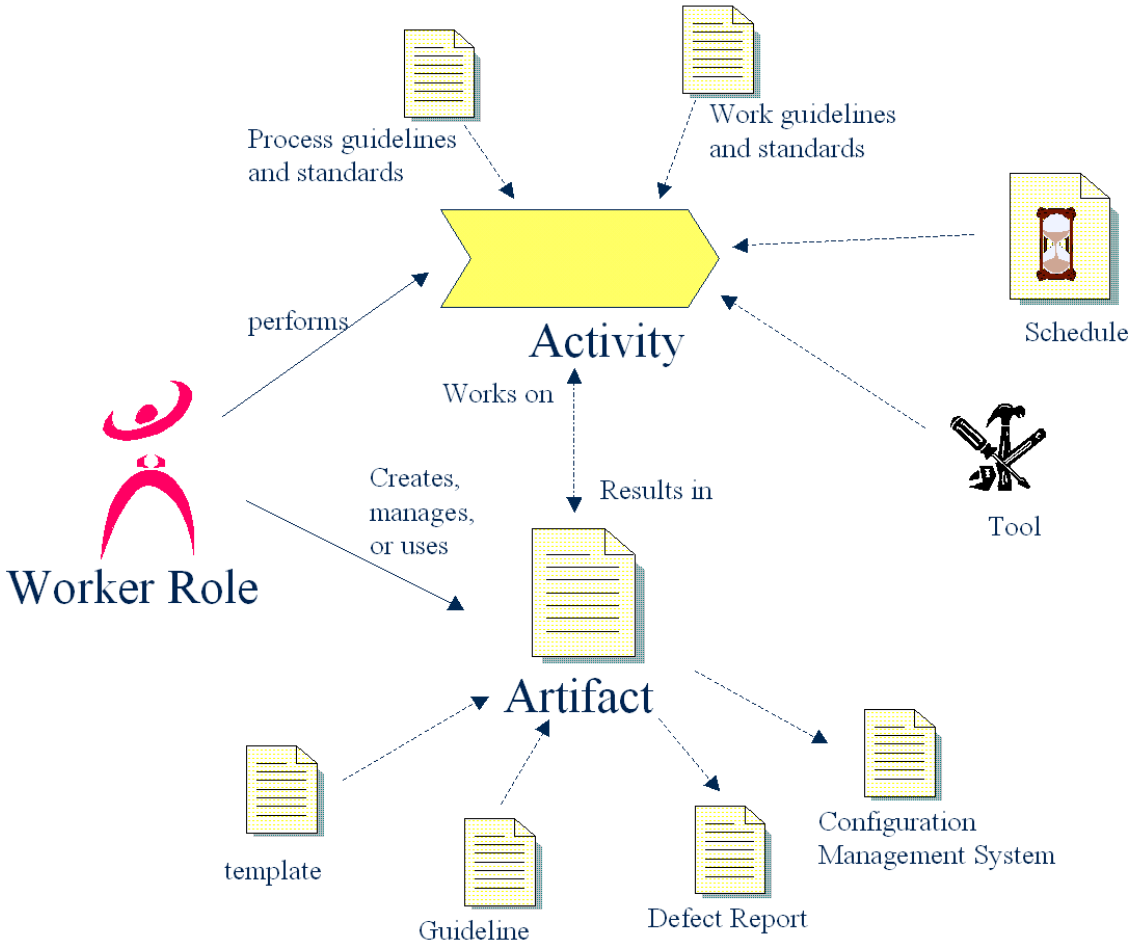

Рис. 2-1: Базовые элементы процесса

Деятельности - это задачи, которые выполняет участник процесса при выполнении своих служебных обязанностей. Деятельности группируются в последовательности работ, имеющие конкретное назначение:

- работы, выполняемые на фазе разработки;
- работы, выполняемые для достижения определенной цели;
- работы, выполняемые по созданию определенного артефакта;
- работы, выполняемые определенной рабочей ролью

Как правило, процесс разбивается на *фазы*, которые могут рассматриваться как деятельности самого крупного масштаба. Каждая фаза определяется с помощью одного или нескольких последовательностей работ. Каждая последовательность работ определяет упорядоченный набор деятельностей – простых задач, выполняемых участниками процесса, и результирующие артефакты. Для отображения последовательностей работ часто используются диаграммы деятельности UML; мы также будем придерживаться данного подхода.

Артефакты – это то, что создается или изменяется при выполнении определенных деятельностей в рамках процесса. Наиболее значимым артефактом является разрабатываемая система; однако существует множество других артефактов – таких как, исходный код, модель программного обеспечения, спецификация требований, тест-векторы, и т.д. Вообще говоря, результатом каждой деятельности является создание или изменение по меньшей мере одного артефакта.

Процесс Harmony, рассматриваемый более подробно в следующем разделе данной главы, применим (и используется в настоящее время) в проектах, имеющих широкий диапазон масштабов. Такая масштабируемость процесса Harmony достигается за счет использования двух различных способов. Во-первых, сам процесс рассматривается на нескольких временных масштабах: макро, микро и нано. В небольших проектах большее внимания уделяется микро- и нано-циклам; однако по мере увеличения размеров проектов, все больше внимания уделяется макро уровню, в котором производится организация и управление всем процессом разработки в целом. Во-вторых, некоторые артефакты не являются обязательными и создаются в процессе работы по мере необходимости. Так, например, анализ потенциальных угроз выполняется только при разработке приложений, предъявляющих особые требования к безопасности. В качестве другого примера можно привести представление архитектуры системы для ее декомпозиции на подсистемы: оно используется только тогда, когда разрабатываются достаточно большие системы, для которых такая декомпозиция будет оправданной. На самом деле процесс Harmony имеет две разновидности: первая, называемая полным процессом Harmony, включает в себя детальный процесс системного инжиниринга, предшествующий разработке отдельных подсистем, и вторая, имеющая название Harmony-ПО, ориентирована исключительна на разработку программного обеспечения.

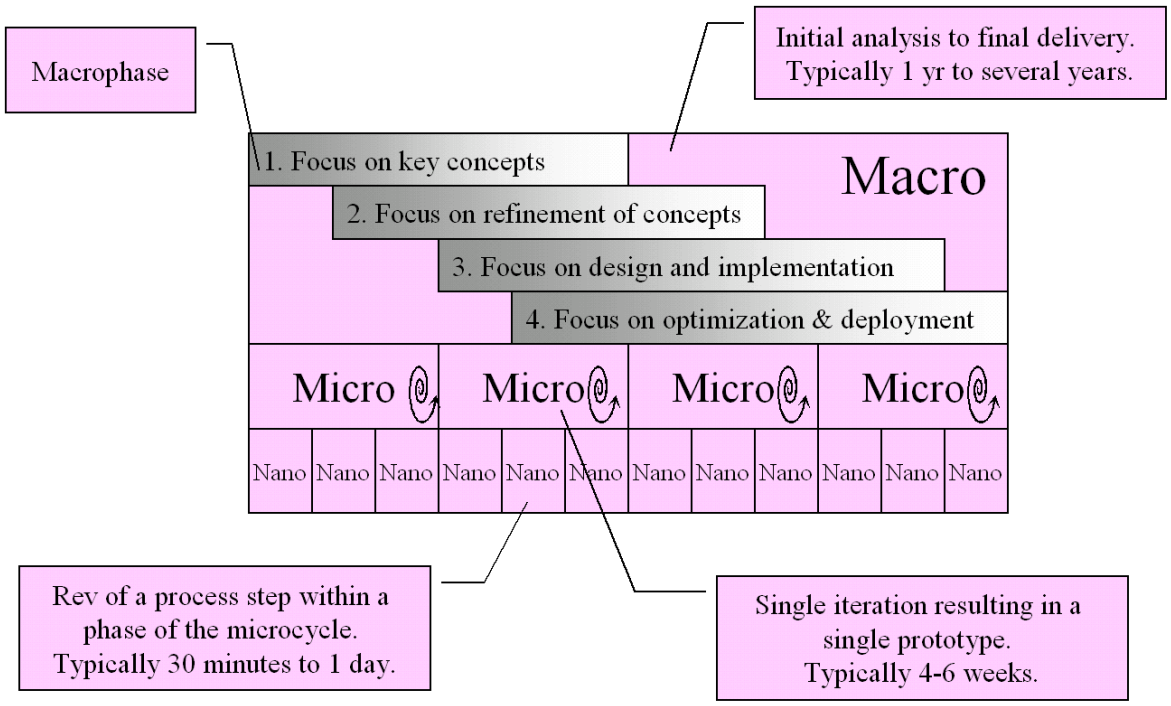

Рис. 2-2: Временные рамки процесса Harmony

Несмотря на очевидные недостатки, модель водопада для жизненного цикла до сих пор широко применяется в планировании и управлении проектами. Но наиболее серьезным недостатком модели водопада является то, что дефекты, возникающие на ранних стадиях процесса не могут быть обнаружены и (или) исправлены до самых последних стадий. Исправление определенных видов стратегических ошибок – таких как ошибок в требованиях и архитектуре – в 3-4 раза дороже в модели водопада, поскольку их влияние распространяется на многие аспекты разрабатываемой системы. Этот недостаток присущ именно модели водопада, поскольку в этой модели *тестирование выполняется в самом конце процесса*. Чем больше времени пройдет перед тем как вы обнаружите и исправите ошибки, тем более они укоренятся в системе и тем большее число ее элементов будет зависеть от ошибочных аспектов. Проблема с моделью водопада заключается в предположении, что при завершении каждого этапа процесса в системе будут отсутствовать более или менее серьезные ошибки, что на самом деле весьма далеко от реальности. Когда ошибки обнаруживаются и исправляются, цена таких исправлений бывает очень высокой.

Спиральная модель жизненного цикла (также известная как итеративная) приобрела популярность благодаря тому, что она устранила недостатки, присущие модели водопада. Основное преимущество спиральной модели состоит в том, что тестирование в ней начинается гораздо раньше и выполняется гораздо чаще. Это позволяет выявлять и исправлять ошибки гораздо раньше и с гораздо меньшими затратами. По сути, спиральная модель разбивает проект по разработке программного обеспечения на несколько более мелких проектов, в которых новые возможности

системы добавляются итеративно, но не прежде, чем уже реализованные возможности пройдут проверку. Каждое добавление нового набора возможностей называется "витком спирали" или "итерацией". Каждый из таких подпроектов имеет гораздо более узкие границы, для их реализации требуется гораздо меньше усилий, и они более сфокусированы на цели, более конкретной чем у всей системы. Результат каждой итерации в процессе Harmony называется *итерационным прототипом* -- это работающая высококачественная система, которая может быть не настолька полна (либо точна), как конечная система. Тем не менее, прототип безошибочно реализует и исполняет некоторую часть требований и/или позволяет снизить выбранные риски и содержит реальный код, который войдет в состав результирующего продукта.

Процесс Harmony можно концептуально рассматривать протекающим в трех различных временных масштабах одновременно (см. Рис. 2-2). Процесс на уровне *макроцикла* обычно длится от многих месяцев до нескольких лет и управляет разработкой в целом -- от начальной концепции до конечной поставки. Макропроцесс Harmony включает четыре основных, но перекрывающиеся, фазы. Как мы скоро увидим, каждая макрофаза в действительности содержит множество микроциклов[2], результатами которых является создание итерационных прототипов.

Макрофазы – это способ отобразить, что назначение создаваемых прототипов изменяется с течением времени в соответствии с определёнными правилами. Ранние прототипы обычно фокусируются на ключевых концепциях, таких как требования, архитектура или технология. Последующие несколько прототипов включают и фокусируются на дополнительных аспектах требований, архитектуры и технологий. После этого фокус смещается на вопросы проектирования и реализации. И, наконец, последние прототипы фокусируются на оптимизации и размещении (на целевом устройстве и в реальном рабочем окружении заказчика). Смещение фокуса прототипов происходит постепенно, отсюда и возникает перекрытие макрофаз. При необходимости, в процесс легко интегрируются предварительные и критические ревизии проекта. Как правило, предварительная ревизия проекта выполняется при приближении к концу первой макрофазы, а критическая ревизия проекта – при приближении к концу второй макрофазы.

Каждый макроцикл содержит несколько микроциклов (или спиралей). Продолжительность каждого микроцикла невелика и как правило составляет 4 – 6 недель. Целью каждого микроцикла является создание и выпуск одного итерационного прототипа с ограниченной функциональностью, но реализованной с высоким качеством. Как правило, каждый такой микроцикл сфокусирован на реализации одного или небольшого числа вариантов использования, но также может включать в себя определенные виды деятельности, направленные на снижение рисков.

В пределах одного микроцикла разработчики работают над созданием высококачественных кооперaций объектов, реализующих варианты использования

[2] т.е. спиралей

выбранных для прототипа. В процессе данной работы, постепенно наполняемые кооперации объектов исполняются от десятков до сотен раз. Этот очень короткий цикл, продолжительностью от нескольких минут до нескольких часов (но не больше), называется наноциклом. Если в кооперации участвуют около 100 объектов, то, как показывает опыт, не стоит все 100 объектов собирать вместе и произносить молитву "Господи, помоги сделать так, чтобы она заработала[3]", а вместо этого начать с реализации одного (неполного) объекта и добиться чтобы он заработал отдельно, пользуясь возможностью исполнения модели. Затем добавить второй объект и добиться чтобы они заработали вместе. Затем уточнить объекты или добавить третий объект, и так далее, внося небольшие изменения к возможностям, реализуемым кооперацией, выполняя валидацию (на основе исполнения) каждого небольшого шага добавления. Средства моделирования с возможностью исполнения моделей, такие как Rhapsody™, позволяют сделать этот процесс высокоэффективным. Основная идея наноциклов – делать очень маленькие шаги по добавлению и убеждаться путем исполнения в том, что они выполнены правильно, прежде чем переходить к следующему шагу. Так называемые "гибкие методологии", одной из разновидностей которых является экстремальное программирование в основном фокусируются на масштабе процесса разработки уровня наноциклов.

2.2.2 Обзор процесса Harmony

Примечание: В оставшейся части этой главы речь пойдет о процессе Harmony версии 1.5. Для процесса Harmony применяется политика управления конфигурациями компании IBM; в результате чего пользователи могут быть уверены, что у них имеется согласованный набор внутренне непротиворечивых артефактов. Предполагается, что с течением времени процесс будет изменяться и обновляться.

Процесс Harmony – это универсальный процесс разработки систем, и хотя в нем делается акцент на разработке ПО реального времени и встраиваемого ПО, он включает все шаги, необходимые для разработки универсального ПО и систем. Процесс Harmony успешно использовался как в очень маленьких проектах, с количеством участников от 1 до 3 человек, так и в крупномасштабных проектах, с несколькими сотнями участников. Harmony является очень масштабируемым процессом "средней тяжести", сохраняющим баланс между статическими "тяжеловесными" процессами и "легкими", так называемыми гибкими методологиями, такими как экстремальное программирование, включая в себя преимущества обоих[4].

---

[3]Молитва -- несомненно прекрасная вещь, однако ее использование и влияние на процесс разработки не будут рассматриваться в этой книге. Из своего опыта я могу сказать, что только при помощи молитв качественное программное обеспечение не может быть разработано. ;-)

[4]Масштаб времени уровня "наноциклов" в процессе Harmony соответствуют масштабу, с которым в основном имеют дело гибкие методологии разработки

Для процесса Harmony определены две основных разновидности. Первый предназначен для более крупномасштабных проектов, в которых имеется существенный объем работ по совместной разработке программного и аппаратного обеспечения. Поскольку время создания прототипа для механических и электронных компонентов достаточно велико, для таких проектов очень важно, чтобы перед началом проектирования *все* требования были собраны и осознаны, а общая архитектура системы определена. Поэтому, макроцикл полного процесса Harmony представляет собой гибрид классического "V"-цикла и спиральной модели, как показано на Рис. 2-3

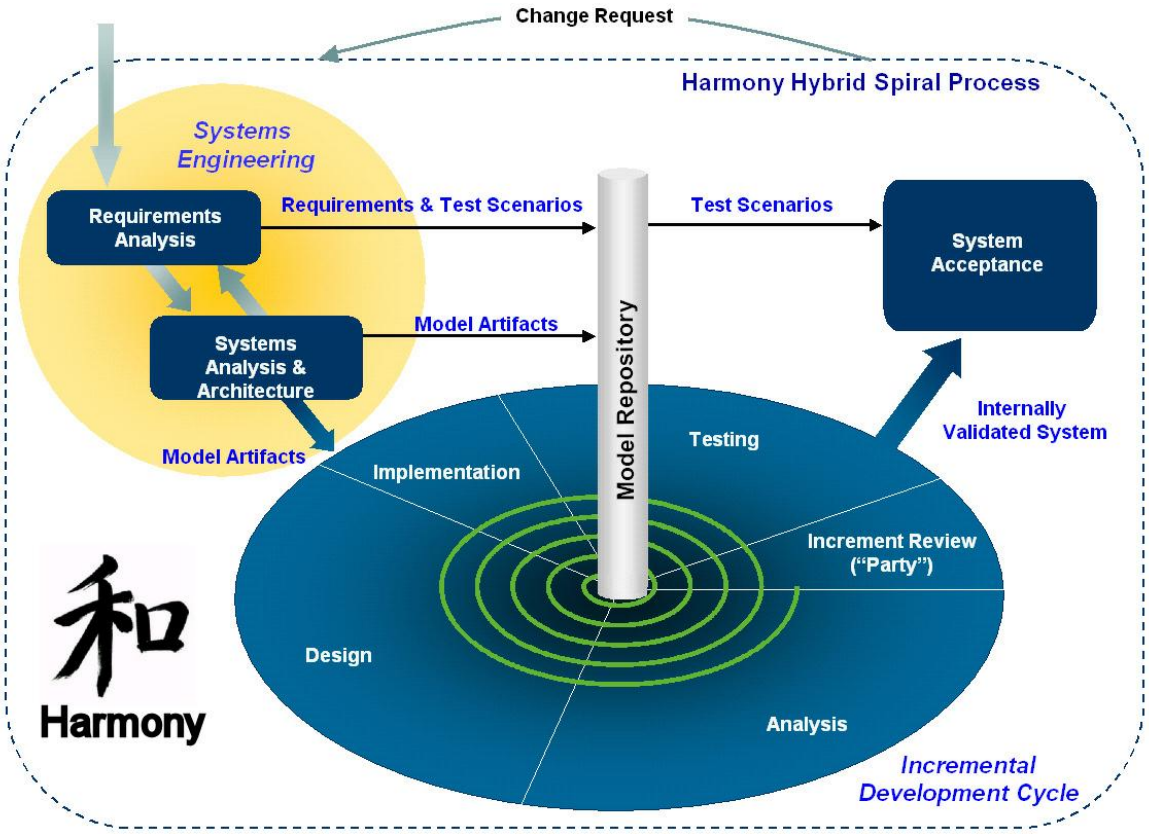

Рис. 2-3: Полный вариант гибридной спирали Harmony

Полный вариант процесса Harmony, изображенный на Рис. 2-3, в начале содержит работы, называемые "системным инжинигингом". В процессе системного инжиниринга полностью определяются требования к системе, требования группируются по вариантам использования, система разбивается на подсистемы, требования привязываются к подсистемам, а затем, на уровне подсистем распределяются по различным инженерным дисциплинам: механическим, электронным, химическим и программным.

Часть процесса, относящаяся в полном варианте гибридной спирали Harmony к системному инжинирингу, была изначально разработана Хансом-Питером Хоффманом

(Dr. Hans-Peter Hoffman), ведущим методологом в области системного инжиниринга компании IBM. Мы в течение нескольких лет работали над созданием полностью интегрированного процесса разработки систем и программного обеспечения; результат нашей работы мы назвали процессом Harmony, поскольку он объединяет дисциплины по разработке систем и программного обеспечения с помощью единого согласованного процесса. Три фазы процесса, относящиеся к системному инжинирингу, показаны на Рис. 2-4

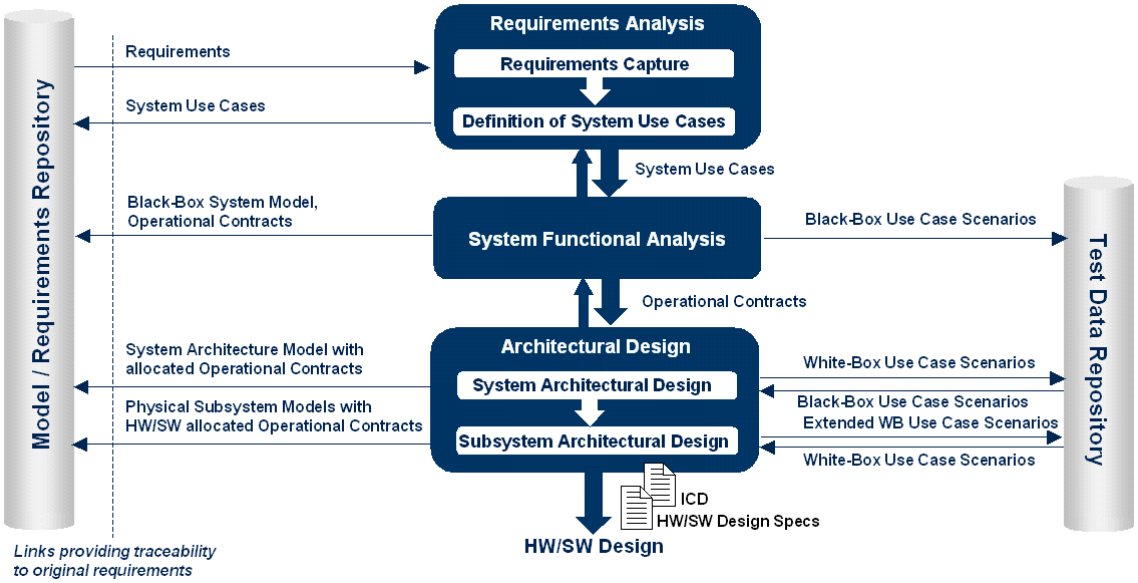

Рис. 2-4: Фазы системного инжиниринга гибридной спирали Harmony

После завершения фаз системного инжиниринга, начинается итерационный цикл разработки. В этой части, два варианта процесса – полный процесс Harmony и Harmony-ПО – очень похожи между собой. Есть одно важное отличие в фазе анализа - в полном варианте процесса на этой фазе выбираются уже предварительно детализированные варианты использования и используются как основа для разработки прототипа, в то время как в варианте процесса по разработке программного обеспечения, варианты использования до этого момента еще не детализированы и должны детализироваться на каждом витке спирали, но в остальном спирали в основном идентичны.

Спиральная часть процесса показана на Рис. 2-5

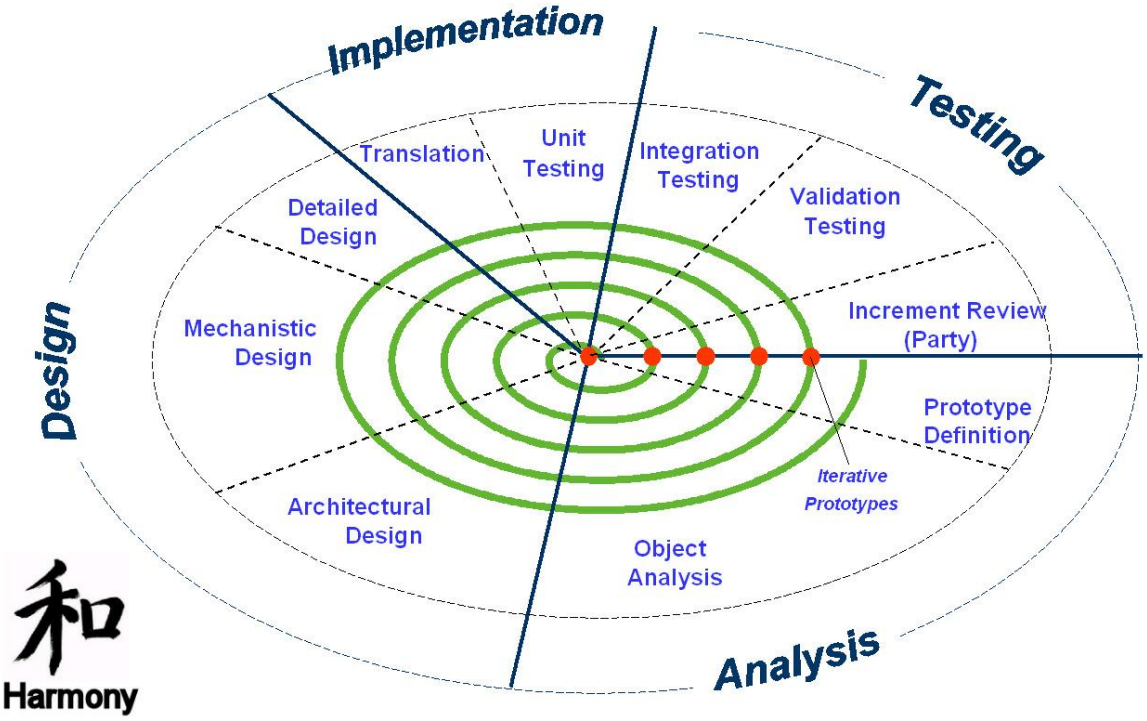

Рис. 2-5: Спираль Harmony (обзор)

В полном процессе Harmony, фаза определения прототипа в процессе анализа состоит лишь в выборе вариантов использования, уже детализированных ранее в процессе системного инжиниринга. В спирали для варианта процесса по разработке ПО, варианты использования только идентифицированы (определено имя и кратко описано назначение), но детальные требования для этих вариантов использования пока еще не определены. В этом случае, в первой части спирали производится детализация вариантов использования, так чтобы они были полностью определены.

В следующих двух разделах будут описываться последовательности работ для каждой из этих фаз. Для отображения деятельностей, их последовательностей и артефактов процесса используются диаграммы деятельности UML.

2.2.3 Детализация последовательностей работ по системному инжинирингу в процессе Harmony

До сих пор микроцикл Harmony был рассмотрен нами только в общих чертах; но для того чтобы понимать как использовать процесс, необходимо более детально какие выполняются в нем виды деятельности и какие получаются артефакты..

2.2.3.1 Функциональный анализ системы

На Рис. 2-6 показана последовательность работ на фазе функционального анализа системы. На рисунке схематично показано, что на данной фазе выбираются варианты использования (по одному или по несколько за раз), и для каждого создается модель варианта использования. Это означает, что создается *исполняемая модель* с использованием семантически полного моделирования, предоставляемого UML. Подробности того как это делается обсуждаться в следующем разделе. Каждый вариант использования оценивается путем исполнения, что позволяет гарантировать его полноту, правильность, согласованность и точность.

Оценка может производиться постепенно, то есть по мере наполнения модели вариантов использования новыми вариантами использования на каждом шаге производится оценка всей модели, либо валидация сначала производится независимо для каждого варианта использования, которые в дальнейшем объединяются вместе. Если используется второй подход, и варианты использования не являются полностью независимыми, то могут возникать ситуации, когда варианты использования будут несогласованными друг с другом. В этом случае анализ согласованность вариантов использования выполняется посредством их объединения в единую модель вариантов использования и исполнения этой модели как целого.

Я хочу обратить Ваше внимание на то, что для представления вариантов использования мы будем использовать объекты, детализируя взаимодействие с ними при помощи диаграмм последовательностей и определяя их поведения при помощи конечных автоматов и/или диаграмм деятельности. Тот факт, что для моделирования мы используем семантически точные языки, *не означает, что мы выполняем проектирование!* Этот момент многими людьми часто понимается не правильно. Использование семантически точного языка, такого как конечные автоматы, не ограничивает нас в том *что* именно мы собираемся сказать -- и лишь означает то, что мы используем семантически точный язык для того чтобы *сказать* это. В нашем контексте мы используем семантически точные языки для спецификации требований, но ничего при этом не говорим о вопросах проектирования или реализации.

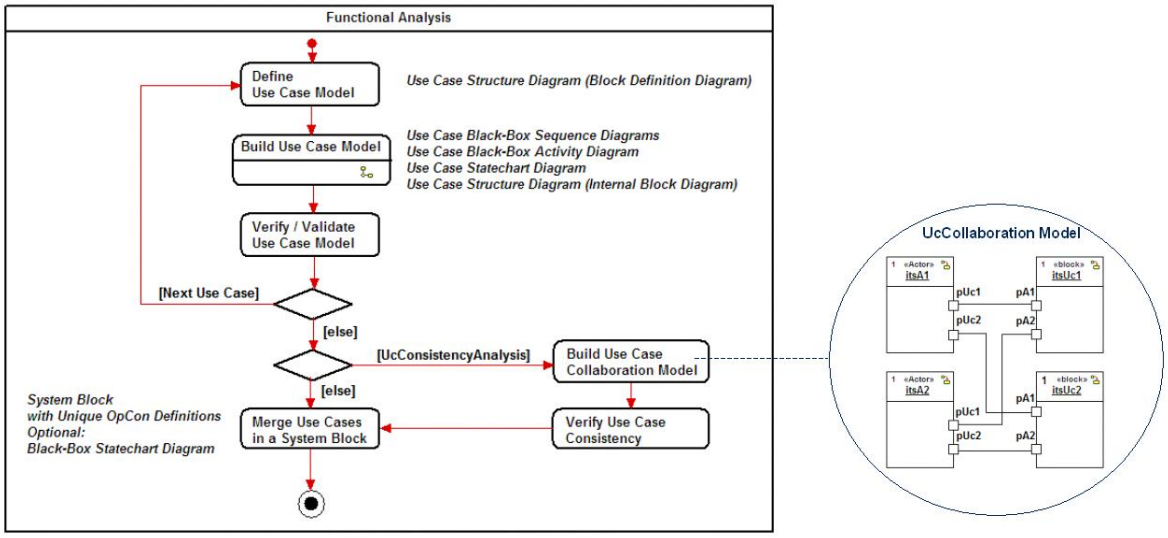

Рис. 2-6: Последовательность работ для фазы функционального анализа системы

2.2.3.2 Последовательность работ для фазы Построение модели вариантов использования

Одним из шагов в предыдущей последовательности работ (Рис. 2-6) было построение модели вариантов использования. Данный шаг детализируется в следующей последовательности работ (Рис. 2-7). Данная последовательность работ отображает три альтернативных подхода, соответствующие различным предпочтениям. На выходе данной последовательности работ вы создаете диаграммы последовательностей, отображающие стандартные и нештатные сценарии взаимодействия вашей системы с ее внешним окружением, диаграмму деятельности, объединяющую в себе все эти сценарии, и диаграмму состояний, определяющую исполняемую модель поведения объекта, который является представлением выбранного варианта использования системы.

Зачем мы используем представление варианта использования в виде объекта? Варианты использования сами являются классификаторами в UML и для них может быть определено поведение, например, с использованием диаграмм состояний. Однако, для вариантов использования не существует возможности отобразить ни интерфейсы, ни порты. Как результат этого, для наших целей мы считаем удобным моделировать варианты использования в виде объектов чисто по техническим причинам. Так как использование объектов удобно только с точки зрения нотации, мы можем думать о нем как о "варианте использования" в другой нотации.

Такой анализ называется анализом "черного ящика", поскольку внутренняя структура системы еще неизвестна или не используется в данный момент. В следующей последовательности работ мы "откроем" этот ящик, определим подсистемы и распределим между ними функции системы.

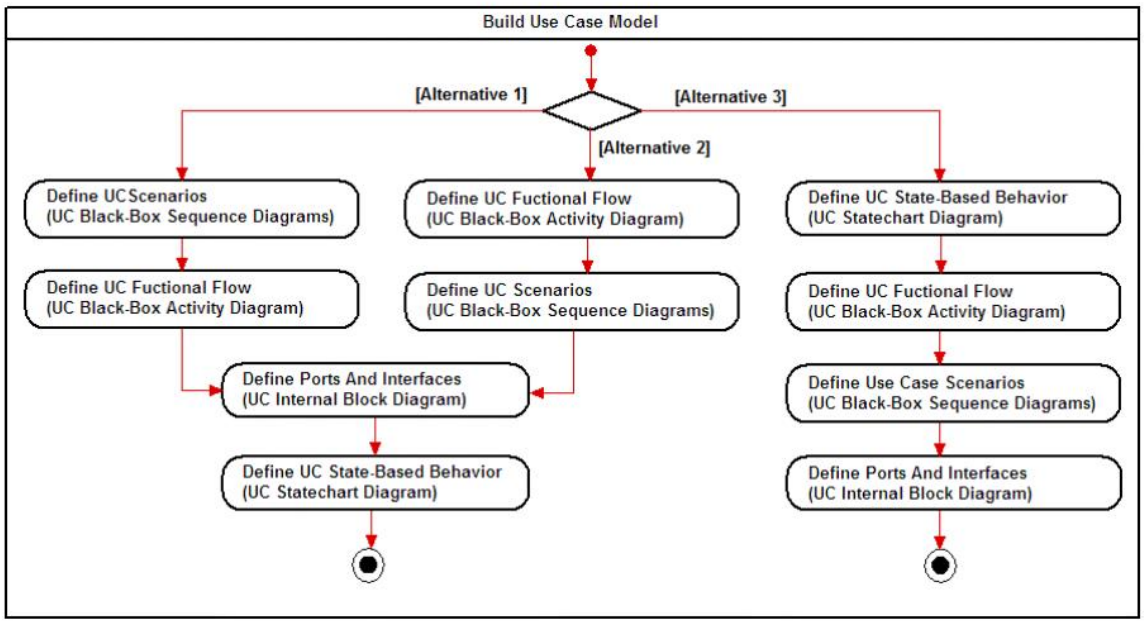

Рис. 2-7: Последовательность работ на фазе Построение модели вариантов использования

2.2.3.3 Проектирование архитектуры системы

Целью следующей последовательности работ является определение общей архитектуры системы. Это достигается путем определения связанных функциональных блоков, представляемых в виде объектов подсистем, а также их точек соединения (портов) и интерфейсов. Операционные контракты распределяются по этим подсистемам. На этой фазе каждая подсистема пока является "междисциплинарной" – т.е. включает в себя элементы различных инженерных дисциплин, таких как электроника, механика и программное обеспечение. Эти подсистемы оцениваются путем их совместного исполнения, в процессе чего показывается что эти подсистемы реализуют те же самые сценарии черного ящика, определенные в предыдущей последовательности работ. Обратите внимание, что сокращение "оп.контр." на рисунке обозначает операционные контракты (определение сервисов в интерфейсах), ЧЯ обозначает "черный ящик", а БЯ – "белый ящик" (т.е. уровень подсистем).

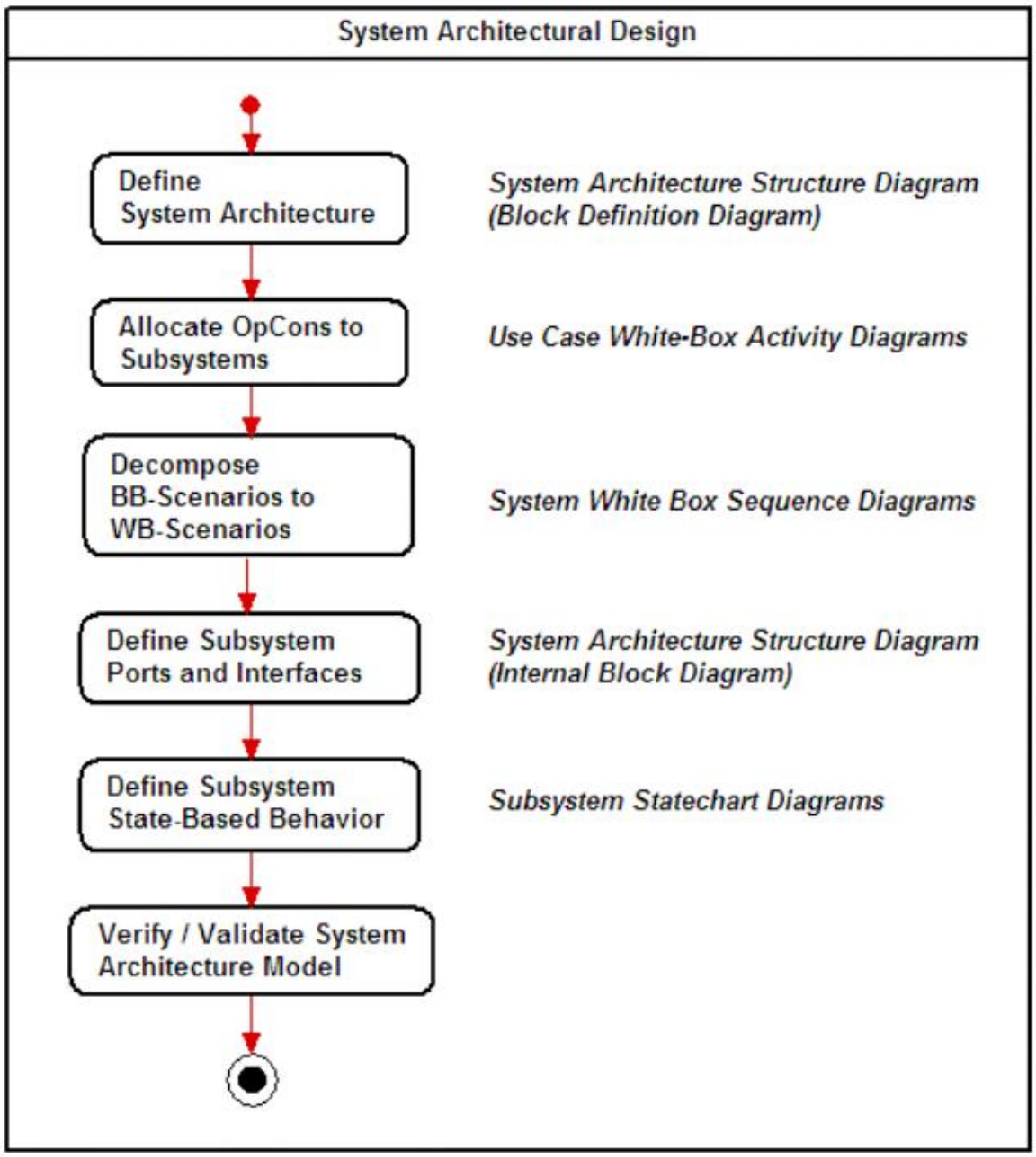

Рис. 2-8: Последовательность работ на фазе Проектирование архитектуры системы

2.2.3.4 Последовательность работ на фазе Проектирование архитектуры подсистем

Последняя последовательность работ, описанная в этом кратком обзоре процесса -- это проектирование архитектуры подсистем (см. Рис. 2-9). Сервисы распределяются по

различным компонентам, относящимся к конкретным инженерным дисциплинам (механическим, электроннным, программным и т.д.). Если какие-то сервисы не могут быть реализованы одной инженерной дисциплиной, то их необходимо декомпозировать (как правило, при помощи диаграмм деятельности) до тех пор, пока это не станет возможным. Интерфейсы между данными компонентами определяются только на высоком уровне, но будут детализированы (например, номера портов или адреса памяти, назначение битов, пред- и пост-условия), когда начнется спиральная часть процесса.

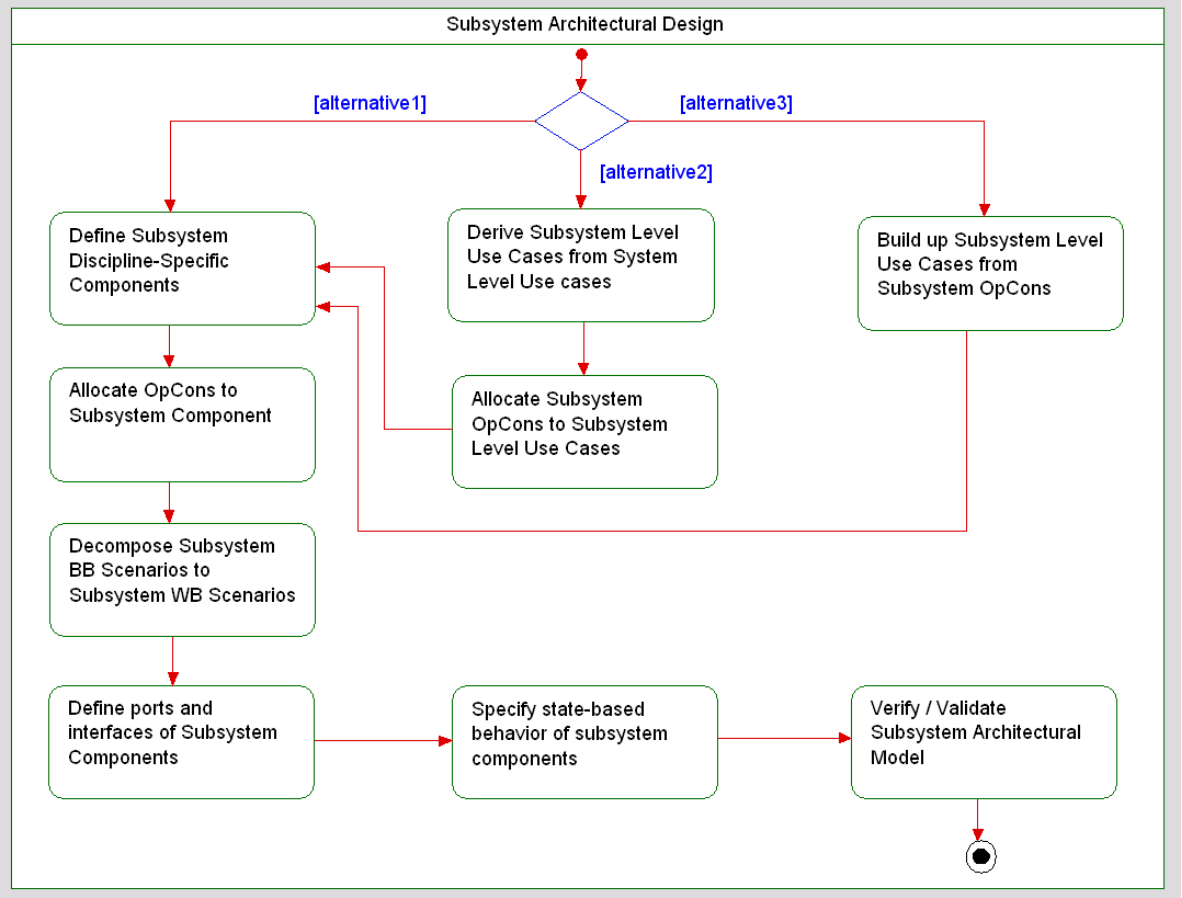

Рис. 2-9: Проектирование архитектуры подсистем

В данной последовательности работ возможно три альтернативных точек входа. В первом случае *не* определяются варианты использования уровня подсистемы, а работа продолжается исходя из определенных ранее операционных контрактов (вариант 1 на рисунке). Во втором случае работа начинается на основе вариантов использования уровня системы, которые декомпозируются с использованием «include»-зависимости. Каждый вариант использования декомпозируется на множество вариантов использования, каждый из которых может быть реализован одной подсистемой. Как правило, при декомпозиции каждого варианта использования уровня системы

получается один или несколько вариантов использования для каждой подсистемы. Данный процесс производится для каждого варианта использования уровня системы. В результате для каждой подсистемы определяются варианты использования, которые они должны реализовать для того чтобы вся системы могла реализовать свои варианты использования. В последнем случае (альтернатива 3) – работа производится в соответствии с подходом "снизу вверх", в котором множество операционных контрактов группируются в связанные единицы (варианты использования). Лично я предпочитаю второй подход, хотя в случае простых подсистем, подход 1 также может подойти. Некоторые инженеры предпочитают третий вариант в случае, когда подсистемы достаточно сложны, чтобы иметь собственные варианты использования, но при этом они не хотят работать "сверху вниз".

2.2.4 Детализация последовательности работ для итерационного (спирального) цикла разработки

По завершению последовательностей работ, определенных в предыдущем разделе, модель передается междисциплинарным командам ответственным за отдельные подсистемы, и начинается инкрементный цикл разработки (также называемый микроциклом или спиралью). В данном разделе мы детализируем последовательности работ спирали только для программного обеспечения, однако читатель должен осознавать что в гибридной спирали полного процесса Harmony, инженеры других дисциплин также работают итерационным образом. Интеграция осуществляется на фазе тестирования спирали, и данная интеграция может вовлекать механические, химические, электронные и программные дисциплины. Не на всех витках спирали придется осуществлять интеграцию с аппаратурой, но часто это будет иметь место. В качестве аппаратуры могут выступать макеты или прототипы на основе проводной пайки для электронных устройств, а также макеты или вручную выполненные механические компоненты; а также опытные образцы электронных и механических компонентов, полученных с завода-изготовителя. Целью является избежание авральных работ по интеграции с аппаратурой в конце проекта и планирование инкрементной интеграции как можно ранее.

2.2.5 Последовательность работ для фазы Ревизия итерации (Вечерника!)

Спираль начинается с фазы ревизии итерации (также известной как "Фаза Вечеринки"[5]). На этой фазе осуществляется первичное планирование проекта и деятельность по оценке текущих дел. Вы должно быть помните, что существуют два варианта процесса Harmony. В его полном варианте при первом переходе к спирали

[5] "Фаза Вечеринки" может соответствовать фазам по разработке начальной концепции, так и разборке полетов в некоторых других моделях процессов. Термин "Вечеринка" используется нами для того чтобы подчеркнуть что на ней выполняется скорее празднование предстоящего успеха, а не разбор полетов после боя с анализом того что было сделано не правильно.

определяется общий график работ, план разработки программного обеспечения, план конфигурационного управления, а также план по повторному использованию (если требуется). В варианте Harmony-ПО определяются масштабы и границы проекта, подходы к разработке, а также определяются варианты использования системы и кратко описывается их назначение. Но в Harmony-ПО варианты использования *не* детализируются -- это делается в спирали на фазе анализа (рассматриваемой в следующем разделе).

В последующих итерациях на данной фазе выполняется оценка проекта и системы на соответствие определенным планам, и при необходимости производится обновление планов. Основные артефакты, оцениваемые на фазе Вечеринки:

- График работ
- Архитектура
- Процесс
- Риски
- Назначение следующего прототипа

Одна из самых серьезных ошибок в управлении проектами – неадекватная оценка и отсутствие корректировки на протяжении проекта. Как отмечают Де Марко и Листер, "Вы не можете контролировать то, что вы не измеряете[6]". Не менее важно использование выполненных измерений для корректировки проекта. По отношению к графику работ, такие корректировки означают перераспределение ресурсов, изменение очередности работ, исключение некоторых работ из плана, уменьшение (или, наоборот, расширение) границ проекта, повышение (или снижение) качества, изменение графика последующих работ и т.д.

Поскольку выбор и реализация хорошей архитектуры очень важны для долгосрочного успеха проекта и продукта, на фазе Вечеринки производится оценка архитектуры по двум основным критериям. Во-первых, необходимо ответить на вопрос, удовлетворяет ли архитектура требования к качеству, которые определяют выбор архитектуры. Во-вторых, нужно понять, достаточно ли хорошо архитектура масштабируется для будущего роста и развития системы. Процесс реорганизации архитектуры называется *рефакторингом* системы. Если проектная команда обнаруживает, что для каждого следующего прототипа необходимо выполнять рефакторинг системы, это означает, что архитектура плохо масштабируется, и необходимо потратить некоторые дополнительные усилия для выбора более масштабируемой архитектуры.

На самых ранних этапах работы над проектом выбирается способ управления проектом – принимается решение какие инструментальные средства будут использоваться, где будут размещаться эти инструменты и их данные, а также как они будут доступны; определяется политика безопасности и процедуры по ревизии артефактов и оценке качества, и т.д. Фаза Вечеринки позволяет повысить эффективность процесса в ходе

[6] DeMarco and Lister *Peopleware: Productive Projects and Teams* New York, New York, Dorset House Publishing Company, 1987

проекта путем активного обнаружения и устранения проблем и возникающих вопросов.

Мой опыт показывает, что самой большой ошибкой, приводящей к провалу проекта, является игнорирование рисков. Для управления рисками рекомендуется в каждом проекте создавать и поддерживать план управления рисками. В данном плане каждый из рисков должен быть идентифицирован и ранжирован, и если это возможно, должна быть определена стратегия избежания риска. В большинстве случаев это будет определение деятельностей, выполняемых на каждом из витков спирали, направленных на выявление, снижение и управление рисками. На фазе Вечеринки производится ревизия для плана управления рисками, в результате которой в план добавляются новые идентифицированные риски.

И наконец, несмотря на то что ранее уже определены план реализации прототипа и график работ, на каждой итерации производится их ревизия и, возможно, корректировка. Эти корректировки в основном относятся к границам прототипа; но даже если корректировка не производится, явная ревизия плана гарантирует что каждый из участников проекта знает что необходимо делать для завершения микроцикла в течении следующих 4-6 недель.

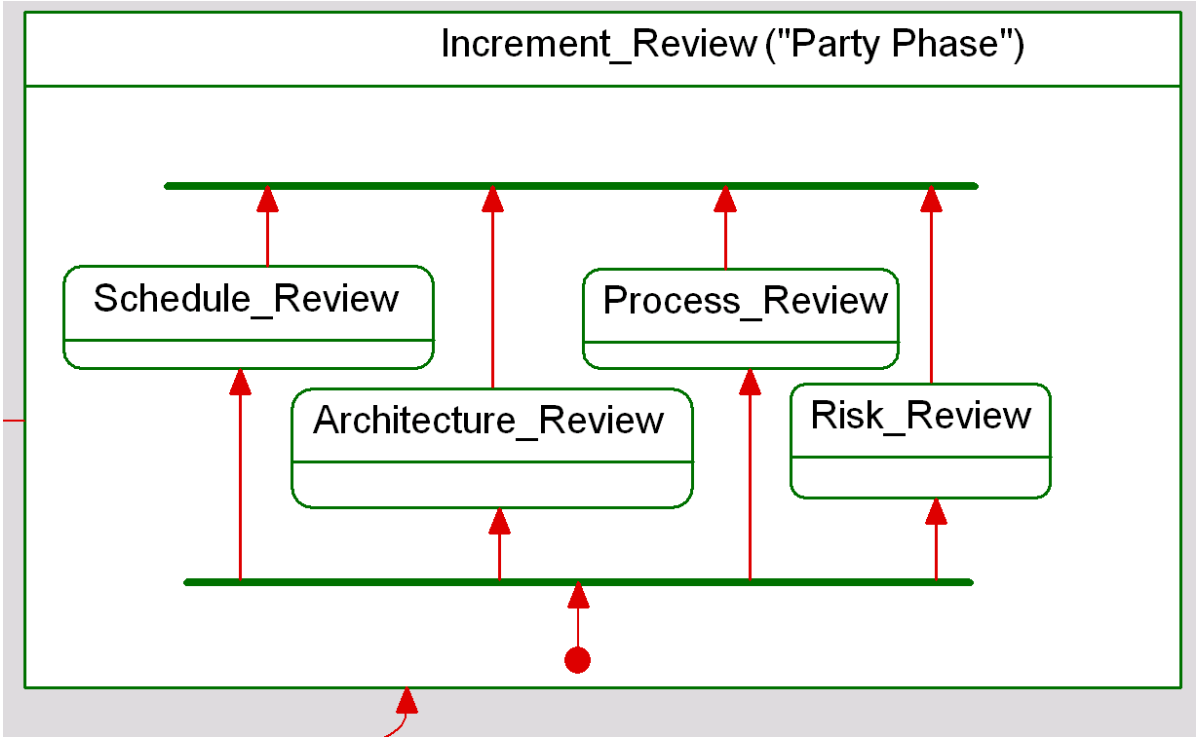

Рис. 2-10: Ревизия итерации (Фаза Вечеринки)

2.2.5.1 Фаза анализ в процессе Harmony

Целью анализа является определение существенных свойств разрабатываемой системы. Термин "существенные" означает, что на данной фазе определяются свойства, отсутствие которых делает систему *незаконченной* или не *несоответствующей* назначению. В терминах Архитектуры на основе моделей (MDA[7]), на фазе анализа мы создаем платформонезависимую модель (PIM) для прототипа. Фаза анализа спирали содержит две основных последовательности работ – определение прототипа и объектный анализ.

2.2.5.2 Последовательность работ по определению прототипа

В полном процессе Harmony эта последовательность работ достаточно тривиальна и сводится к выбору нескольких уже полностью определенных вариантов использования, которые будут включены в прототип на данной итерации. В процессе Harmony-ПО, варианты использования до этого момента уже идентифицированы, но не детализированы; поэтому теперь те варианты использования, которые планируются к реализации в текущем прототипе, необходимо полностью детализировать (те, которые должны быть реализованы в следующих прототипах, в данный момент просто игнорируются). Далее в этом разделе мы сфокусируемся на варианте процесса Harmony-ПО. Последовательность работ для него показана на Рис. 2-11.

На данной фазе детально определяются и фиксируются требования к создаваемому прототипу. Варианты использования для прототипа уже определены, однако детальная спецификация для этих вариантов использования еще не выполнена.

Существует два основных способа для детализации вариантов использования: на основе примера и на основе спецификации. Под детализацией "на основе примера" мы понимаем определение набора сценариев, иллюстрирующих штатные и нештатные взаимодействия для рассматриваемого варианта использования. Преимущество использования сценариев заключаются в том, что они достаточно просты для понимания нетехническими заинтересованными лицами, а также являются основой для тестовых сценариев, которые будут использоваться для проверки реализованного прототипа. К недостаткам использования сценариев относится то, что информация о требованиях связанных с вариантом использования распределена по множеству диаграмм последовательностей (может быть, нескольким десяткам) а не собрана в одном месте, в результате чего требования сложно представить в сжатой, компактной форме. Кроме того, на диаграммах невозможно отразить требования, такие как: "Танк должен иметь камуфляжную окраску в зелено-коричневых тонах", которые не относятся к поведению и являются просто характеристиками, реализуемыми или нереализуемыми в результирующей системе[8]. Для определения сценариев почти всегда используются диаграммы последовательностей UML.

---

[7] Стандарты, связанные с MDA, доступны по ссылке www.omg.org.

[8] Такие требования называются параметрическими требованиями к системе и соединяются трассировочными связями не с вариантами использования, (как это

Во втором способе детализация вариантов использования производится путем их спецификации. Данная спецификация может быть неформальной и основываться на использовании обычного текста для описания требований к варианту использования, или формальной и основываться на использовании формального языка для описания поведения, например, с использованием диаграмм состояний или диаграмм деятельности UML. Преимущества детализации вариантов использования путем спецификации заключается в том, что она более лаконична, может быть выполнена более точно, чем сценарии, а также с помощью них легко представить некоторые требования, которые сложно представить в виде сценариев. Недостатком использования спецификаций является трудность ее понимания нетехническими участниками проекта, а также трудность прямого связывания требований с проетом системы. Если требуется реализовать непрерывное и кусочно-непрерывное поведение, мы рекомендуем использовать диаграммы алгоритмов управления или диаграммы деятельности для представления непрерывного поведения этих отдельных вариантов использования.

Мы считаем, что необходимо использовать одновременно неформальное текстовое описание и формальные языки для детализации вариантов использования. Естественный язык незаменим при объяснении почему это необходимо сделать так, а не по другому , поскольку он богат и выразителен. Но, одновременно естественный язык является также нечетким, допускающим двусмысленные толкования и неточным. Формальные языки превосходят его, когда необходимо сформулировать "что" необходимо сделать. Нилучший результат дает сочетание точного формального описания, например, с помощью диаграммы состояний, и текста с пояснениями.

Оба способа (на основе примеров и на основе спецификации) находят применение, и на самом деле процесс Harmony реомендует использовать их *одновременно*. Формальная спецификация на основе диаграмм состояний или диаграмм деятельности позволяет зафиксировать требования в более компактной форме, а сценарии, полученные из формальной спецификации, могут помочь нетехническим заинтересованным лицам в понимании системы. Позднее, на основе сценариев, полученных из формальной спецификации, могут создаваться тестовые сценарии для оценки в конце микроцикла.

Требования[9] детализируются при помощи комбинации:

- Диаграмм последовательностей
- Диаграмм состояний

делается для функциональных требований и требований к качеству), а с объектом, представляющим всю систему.

[9] В UML отсутствует понятие "требование" в виде основной концепции. С использованием языка SysML, недавно разработанном профиле UML предназначенном для системного инжиниринга, требования представимы в явном виде. Более подробную информацию вы можете найти на сайтах www.omg.org, www.ibm.com.

- Диаграмм деятельности
- Диаграмм алгоритмов управления (не входит в UML)
- Текстовых описаний
- Ограничений к качеству сервиса (Quality of service, QoS)
- Диаграмм требований (языка SysML)

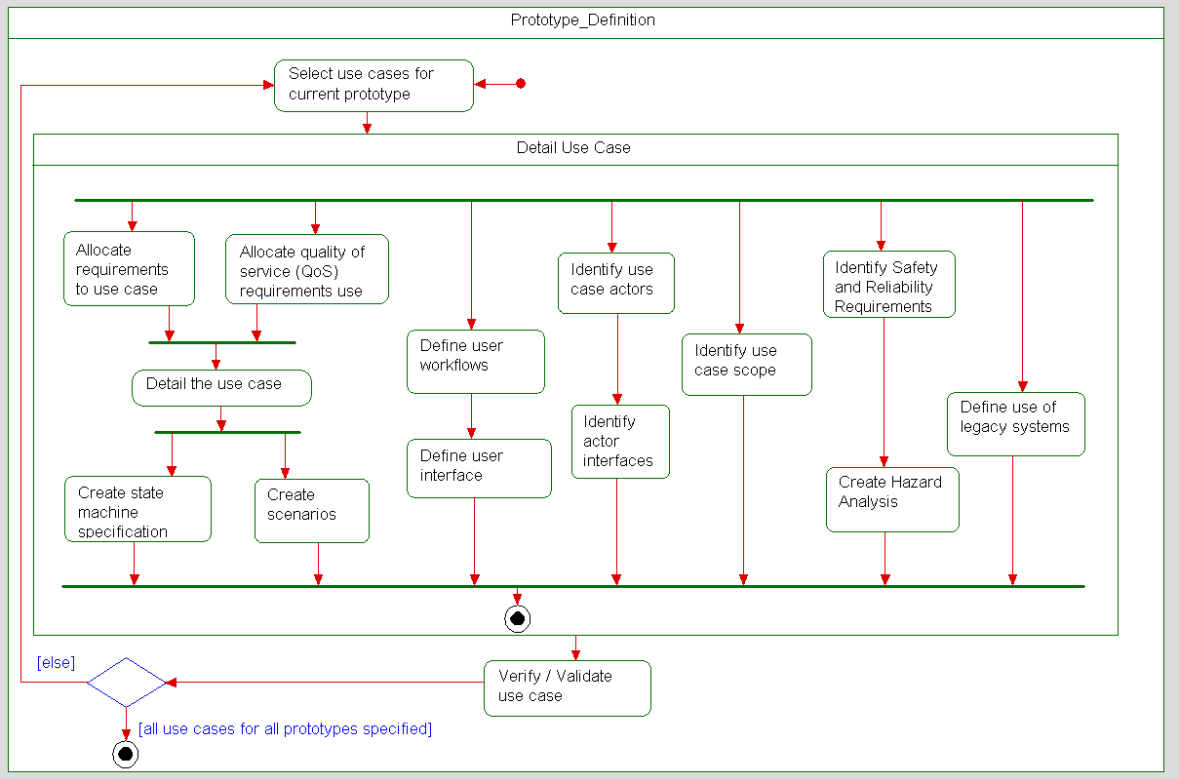

Рис. 2-11: Определение прототипа в процессе Harmony-ПО

2.2.5.3 Фаза объектного анализа

Вариант использования можно рассматривать как некую "емкость", в которой содержится множество детализированных требований, относящихся к данной возможности системы или взаимодействию с нею. . Реализацией варианта использования является кооперация - множество объектов, взаимодействующих друг с другом и соответствующих данному множеству согласованных между собой требований.

На фазе объектного анализа процесса Harmony создаются такие кооперации *существенных* объектов, что производится для одного варианта использования за раз. Это означает, что для текущего прототипа создается по одной кооперации для каждого варианта использования, реализуемого в прототипе. В терминах MDA, существенная модель создаваемая на данной фазе называется платформонезависимой моделью (PIM). В процессе Harmony модель PIM разрабатывается итеративно, путём добавления на

каждой итерации одного или нескольких вариантов использования. Это проиллюстрировано на Рис. 2-12.

В процессе Harmony используются целый набор стратегий по выявлению объектов. Данные стратегии и короткие описания приведены в Таблица 2-1. Мы находим эти стратегии чрезвычайно эффективными для целей выявления существенных классов и объектов в рамках кооперации. Применению этих стратегий посвящена глава 5.

Таблица 2-1: Стратегии выявления объектов

Стратегия	Описание
Подчеркните имена существительные	Для получения первоначального грубого списка объектов, аналитик подчеркивает в формулировке задачи все существительные и словосочетания с существительными, и рассматривает их в качестве потенциальных объектов.
Выявитете объекты, инициирующие действия	Выявите источники действий, событий и сообщений; включите координаторов действий.
Выявите сервисы (предоставляемые пассивно)	Выявите на кого направлены действия, события и сообщения, а также сущности, которые пассивно предоставляют сервисы, когда их запрашивают.
Выявите потоки сообщений и информации	Для сообщений должны быть определены объекты, который их отправляют, и объекты, который их принимают, а также, возможно, другие объекты, которые обрабатывают информацию, которую содержит сообщение.
Выявите объекты реального мира	Объекты реального мира – это сущности, которые существуют в реальном мире, но совсем не обязательно являются электронными устройствами. Например, к ним могут относиться газы, давления воздуха, силы, анатомические органы, химические элементы и субстанции, цистерны и баки, и т.д.
Выявите физические устройства	К физическим устройствам относятся датчики и приводы системы, а также электронные устройства которые ими управляют или контролируют. Во внутренней архитектуре системы это могут быть процессоры или дополнительные электронные блоки. Это особая разновидность предыдущей стратегии.
Выявите основные концепции	Ключевые концепции могут моделироваться как объекты. Банковские счета существуют только в виде концепций, однако являются важными объектами в банковской области. Элементы разрешения по частоте для корреляционного приемника также могут

Стратегия	Описание
	являться объектами.
Выявите транзакции	Транзакции – это механизмы по реализации взаимодействий между обектами, которые имеют место на протяжении некоторого значительного промежутка времени. Примеры транзакций: сообщения шины или данные очереди.
Выявите хранимую информацию	Объекты или атрибуты могут использоваться для представления информации, которая должна сохраняться в течение значительных промежутков времени. Время хранения может превышать время работы между перезапусками устройства.
Выявите визуальные элементы	Элементы пользовательского интерфейса, которые отображают данные, являются объектами из домена пользовательского интерфейса; к их числу относятся окна, кнопки, линейки прокрутки, меню, гистограммы, временные диаграммы, пиктограммы, битовая и растровая графика, а также шрифты.
Выявите элементы управления	Элементы управления – это объекты, предоставляющие пользователю (или некоторому внешнему устройству) интерфейс для управления поведением системы.
Примените сценарии	Просмотрите сценарии, в которых задействованы выявленные объекты. Вы легко обнаружите пропущенные объекты, когда существующие объекты не смогут предоставить требуемых действий.

В процессе анализа необходимо быть аккуратным, чтобы минимизировать количество элементов системы, определенных в процессе анализа . Ограничьте на данной фазе кооперацию лишь теми элементами, которые без всякого сомнения должны присутствовать в модели объектного анализа. Например, для случая, когда вы описываете кооперацию для варианта использования "Управление счетом" в банковской информационной системе, отсутствие в кооперации таких объектов, как *Клиент*, *Счет*, *Дебетовая транзакция* и *Кредитная транзакция* будет точно *не правильно*. В навигационной системе вы ожидаете увидеть такие концепции, как *Положение*, *Направление*, *Тяга*, *Скорость*, *Высота*, *Точка маршрута* и *Траектория*, представленые в виде объектов или их атрибутов. Ваша цель – включить только те объекты, классы и отношения, которые существенны для корректности системы, и отложить оптимизацию до фазы проектирования.

Основной вопрос задаваемый при определениии кооперацийобъектов звучит так: Это правильно? Все ли концепции должным образом представлены? Правильно ли определены отношения между этими концепциями? Является ли их поведение соответсвующим? Отеты на эти вопросы могут быть быстро получены в течении

наноцикла. Деятельность в рамках наноцикла для спирали Harmony показана на Рис. 2-12. Идея состоит в том чтобы производить небольшие постепенные изменения и затем как можно быстрее исполнять кооперацию для того чтобы убедиться что все было сделано правильно. Единственный способ проверки корректности объектной модели - это запустить её на *исполнение и протестировать*. Используя среду разработки с возможностью исполнения моделей, такую как Rhapsody, это может быть сделано очень легко и быстро.

На каждом наноцикле производится наполнение и исполнение модели объектного анализа, которая находится в различных стадиях готовности, не дожидаясь конца разработки. Тестирование становится непрерывным процессом, а не деятельностью, которую необходимо выполнить в самом конце. Это приводит к созданию систем более высокого качества, меньшими усилиями и за более короткое время. Возьмите диаграммы последовательностей, которые отображают сценарии детализирующие требования, *конкретизируйте* сценарии добавив к ним только что определенные объекты и покажите путем исполнения, что данные объекты выполняют отведенные им роли при реализации данных сценариев. Это является ключевой концепцией гибких (agile) методологий разработки, таких как экстремальное программирование: делать как можно более мелкие шаги и оценивать их прежде, чем двигаться дальше.

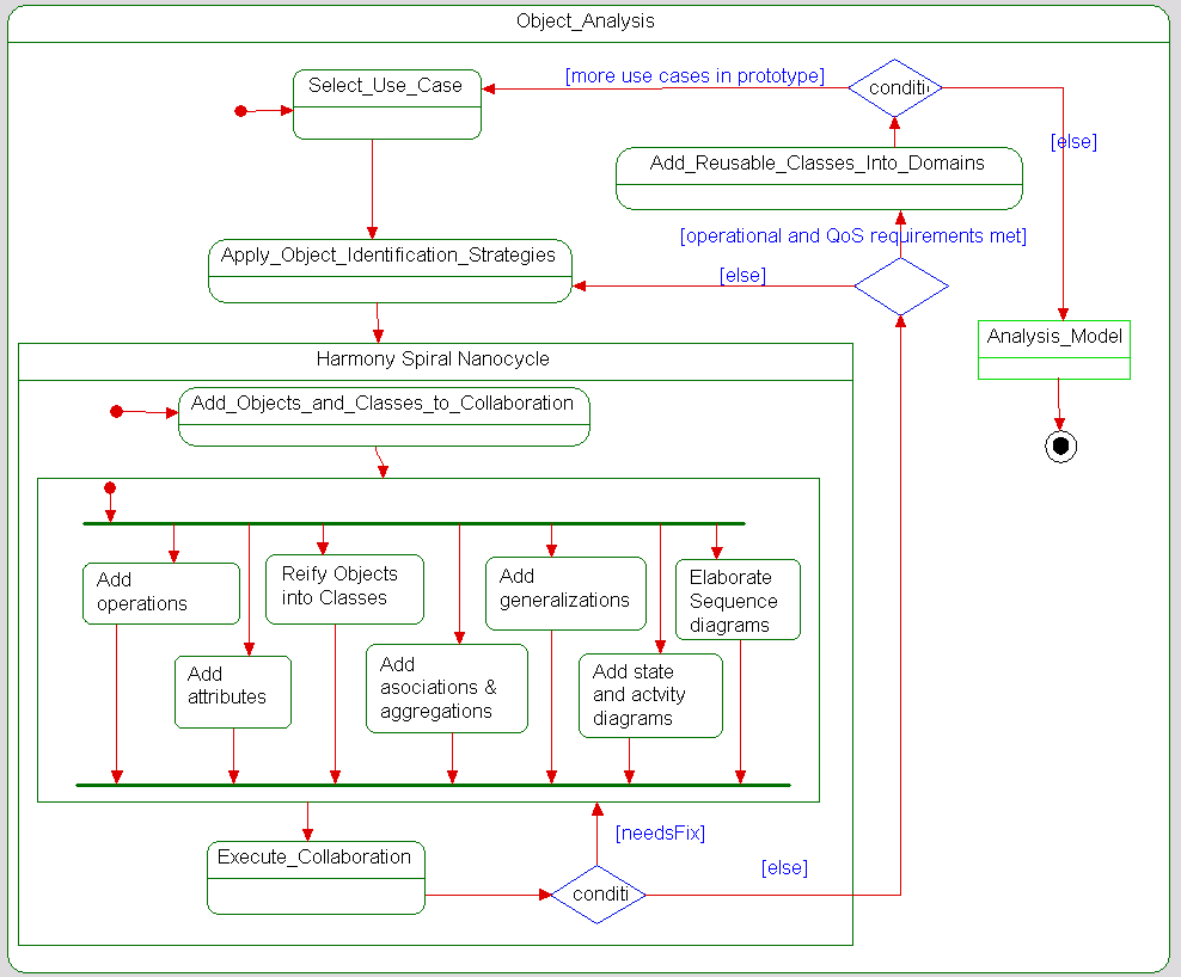

Рис. 2-12: Последовательность работ для объектного анализа

2.2.6 Фаза проектирования в процессе Harmony

Модель анализа определяет согласованный набор свойств, которыми должна обладать разрабатываемая система. Они представлены в виде множества вариантов использования и их детализации (с помощью диаграмм последовательностей, диаграмм состояний, диаграмм деятельности, ограничений на качество сервисов), а также кооперации существенных объектов, чья правильность проверена на основе исполнения и тестирования. При создании модели объектного анализа исходят из функциональных требований и демонстрируют ее функциональную корректность. Как правило, такая модель *не* является оптимальной. Оптимизация модели выполняется в ходе проектирования.

Проектная модель -- это точный чертеж, определяющий *как* будут реализованы существеные свойства системы. При проектировании модель анализа может быть реализована множеством различных способов, на основе различных критериев для оптимизации. В то время как модель анализа определяет множество возможных вариантов решения, модель проектирования является конкретным решением задачи.

Результаты проектирования всегда являются оптимизацией модели анализа. Процесс проектирования, включает в себя следующее:

- Определение критериев проектирования (оптимизации)
- Ранжирование критериев проектирования по их критичности.
- Определение паттернов проектирования или других решений, которые оптимизируют наиболее важные критерии проектирования за счет менее важных.

Многие критерии проектирования являются ограничениями по качеству, определенными на фазе анализа требований. Могут также существовать другие критерии, например, уровень надежности и безопасности, возможность повторного использования, возможность поддержки, простота, время вывода на рынок, и т.д.

В процессе Harmony для проектирования рассматриваются три уровня абстрагирования. Архитектурный уровень абстракции оптимизирует систему в целом на самом верхнем уровне. Как мы увидим чуть позже, существует пять (иногда шесть) архитектурных представлений, которые могут быть оптимизированы более или менее независимо друг от друга. На техническом уровне абстрагирования фокусируются на оптимизации уровня кооперaций, реализующих отдельные варианты использования. Область интересов в данном случае на порядок уже, чем для архитектурного уровня . И наконец, детальное проектирование выполняется на уровне отдельного класса или объекта. На этом уровне оптимизация проводится для отдельных объектов, при этом внимание концентрируется на 5% (или около того) объектов, для которых определены особые требования.

2.2.6.1 Фаза архитектурного проектирования

Как упоминалось ранее, процесс Harmony выделяет пять (или шесть) важных архитектурных представлений.

- Представление подсистем и компонентов
- Представление параллелизма и ресурсов
- Представление развертывания
- Представление безопасности и надежности
- Представление размещения системы
- (опционально) Представление защищенности

На фазе архитектурного проектирования спирали уточняется один или несколько из этих представлений, в соответствии с потребностями текущего прототипа, В основном это делается путем применения архитектурных паттернов проектирования (в моей книге *Real-Time Design Patterns* [10] детально рассматриваются паттерны для каждого архитектурного представления). Эти паттерны достаточно масштабны, и влияют на большую часть или на всю систему в целом.

[10] Douglass, Bruce Powel *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems* (Addison-Wesley, 2002).

Для представлений архитектурного проектирования используются те же диаграммы UML, что и для представлений архитектуры системы и объектного анализа – диаграммы классов используются для представления структуры, диаграммы состояний используются для описания поведения отдельных элементов, а диаграммы последовательностей – для описания взаимодействия внутри групп таких элементов.

Представление подсистем и компонентов определяет архитектурные элементы системы большого масштаба, их назначение, а также интерфейсы между элементами и интерефесы с внешними действующими лицами. Представление параллелизма и ресурсов определяет параллельно исполняемые единицы (моделируемые как "активные" объекты), политики диспетчеризации, а также как они синхронизуются и совместно используют ресурсы. Представление развертывания определяет, как объекты размещаются по различным адресным пространствам, как они находят друг друга, а также средства и протоколы, используемые ими для взаимодействия и совместной работы. Представление размещения отображает каким образом продукты различных инженерных дисциплин работают вместе. Несмотря на то, что UML предоставляет специальные диаграммы размещения, в SysML для этих целей предпочитают использовать диаграммы классов, т.к. эти диаграммы более выразительны, нежели диаграммы размещения. И, наконец, представление безопасности и надёжности играет важную роль для некоторых систем, и должно определять политики и процедуры обеспечения целостности данных и безопасности.

На Рис. 2-13 показана последовательность работ для архитектурного проектирования. На рисунке показано, как определяются критерии проектирования, для них назначаются приоритеты, и затем используются для выбора подходящих архитектурных паттернов проектирования. Параллельность применения паттернов проектирования для различных аспектов архитектуры подчеркивает, что: 1) порядок, в котором они вводятся, выбирается исходя из личных предпочтений, и 2) не все аспекты архитектуры должны быть реализованы в одном прототипе. Обычно аспекты архитектуры обладающие низкими рисками откладываются на более поздние прототипы, а на ранних прототипах концентрируются на аспектах с высокими рисками.

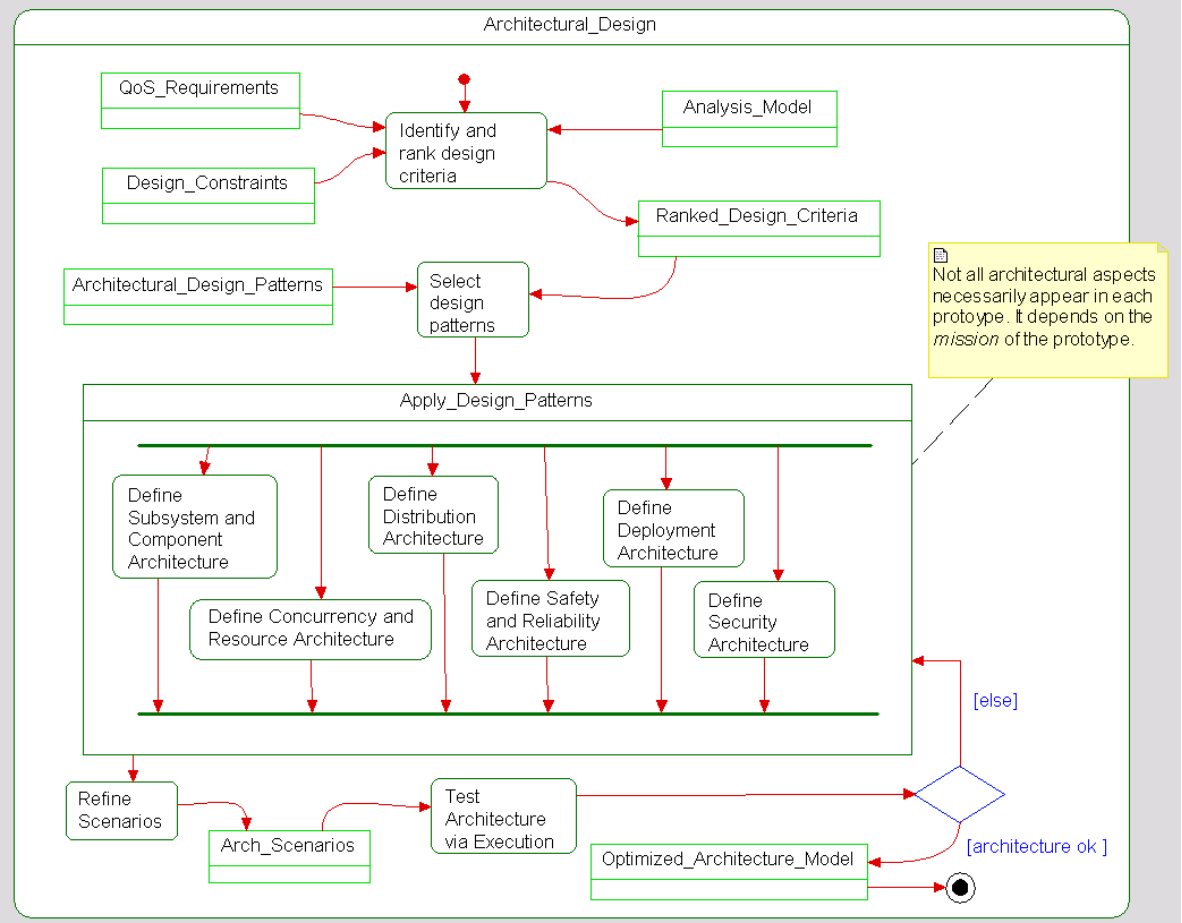

Рис. 2-13: Последовательность работ для архитектурного проектирования

2.2.6.2 Фаза технического проектирования

На фазе технического проектирования производится оптимизация отдельных кооперaций, каждая из которых реализует один вариант использования. Границы для технического проектирования, как правило, на порядок уже, чем для архитектурного проектирования, поскольку система обычно содержит от одного до нескольких десятков вариантов использования. Аналогично архитектурному проектированию, техническое проектирование в большинстве своем производится путем применения паттернов проектирования, хотя границы паттернов гораздо уже, чем для архитектурного проектирования. Именно здесь применяются классические паттерны "банды четырех"[11] и другие более специализированные паттерны.

Модель технического проектирования уточняет модель объектного анализа и использует те же графические представления - диаграммы классов и диаграммы

[11] Gamma, et al *Design Patterns: Elements of Reusable Object Oriented Architecture* (Addison-Wesley, 1995)

последовательностей для представления структуры и взаимодействия, диаграммы деятельности и диаграммы состояний для описания поведения.

Последовательность работ, выполняемых при техническом проектировании, показана на Рис. 2-14. Она очень похожа на последовательность работ при архитектурном проектировании. На первом шаге определяется что будет оптимизироваться (т.е. критерии проектирования), насколько важен каждый из критериев, и далее производится выбор паттернов проектирования, которые их оптимизируют, за счет наименее важных. Полученая в результате проектирования кооперация объектов подвергается тестированию – но не только для проверки что в исходную кооперацию (с фазы анализа) не были внесены ошибки, но и для проверки, что вы добились необходимого уровня оптимизации.

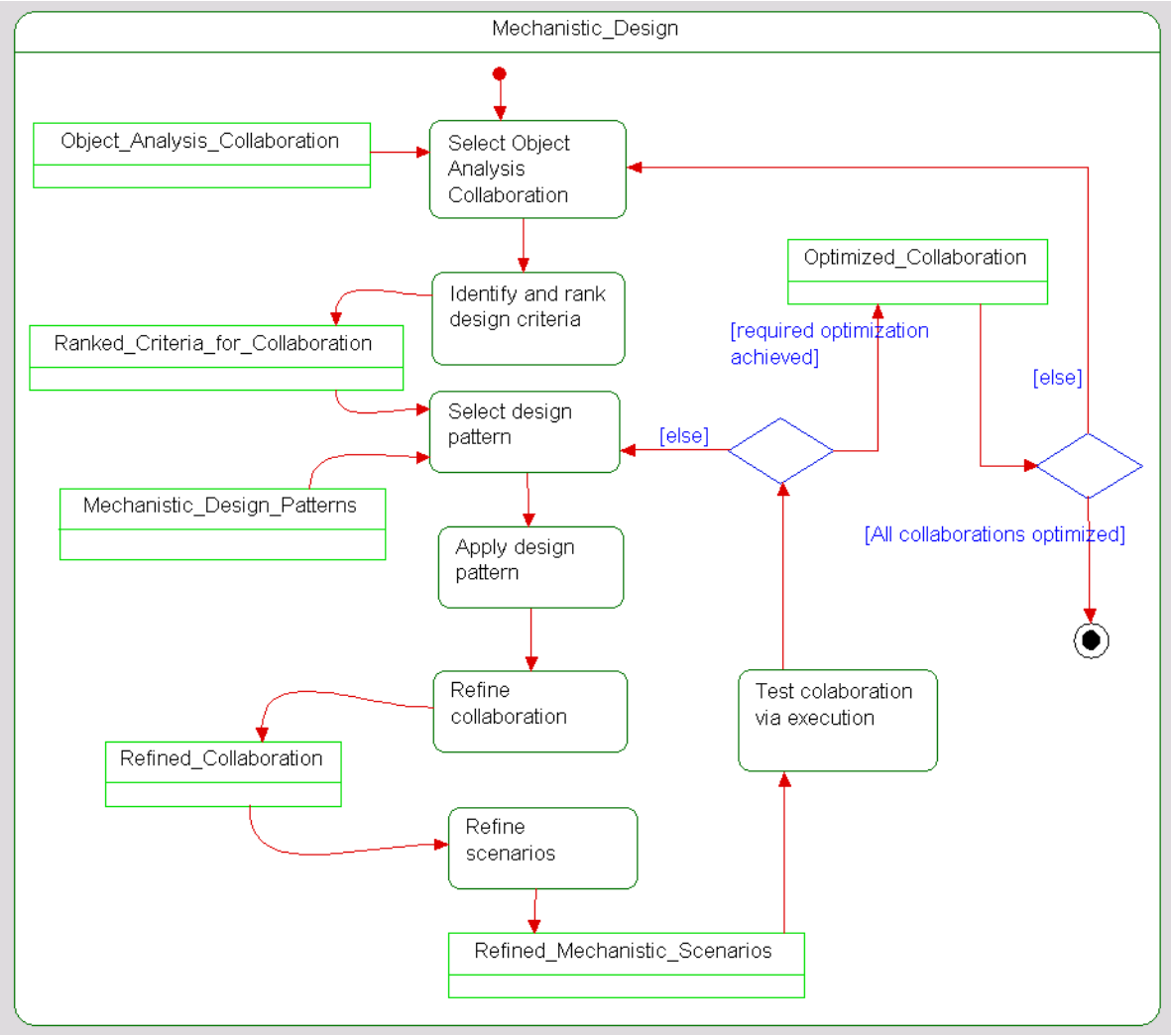

Рис. 2-14: Последовательность работ для технического проектирования

2.2.6.3 Фаза детального проектирования

Фаза детального проектирования предназначена для тщательной проработки реализации объектов и классов и имеет очень узкие границы интересов – отдельный объект или класс. Оптимизация при детальном проектировании в основном касается следующих вопросов:

- Структурирование данных (оптимизация размеров и скорости доступа)
- Алгоритмическая декомпозиция
- Оптимизация конечного автомата объекта
- Стратегии реализации объекта
- Реализация ассоциаций
- Видимость и инкапсуляция
- Проверка во время исполнения пред- и пост- условий (например, для диапазонов параметров, передаваемых в методы).

Существует множество эмпирических правил, рекомендаций и практик для детального проектирования, хотя большинство из них подпадает под определение "идиом", а не "паттернов". Для большинства объектов детальное проектирование не очень сильно отличается от тривиальной детализации; однако, обычно находится относительно небольшое (как правило, 5%), но очень важное множество объектов, которые требуют особого внимания при детальном проектировании. На Рис. 2-15 показана последовательность работ для детального проектирования.

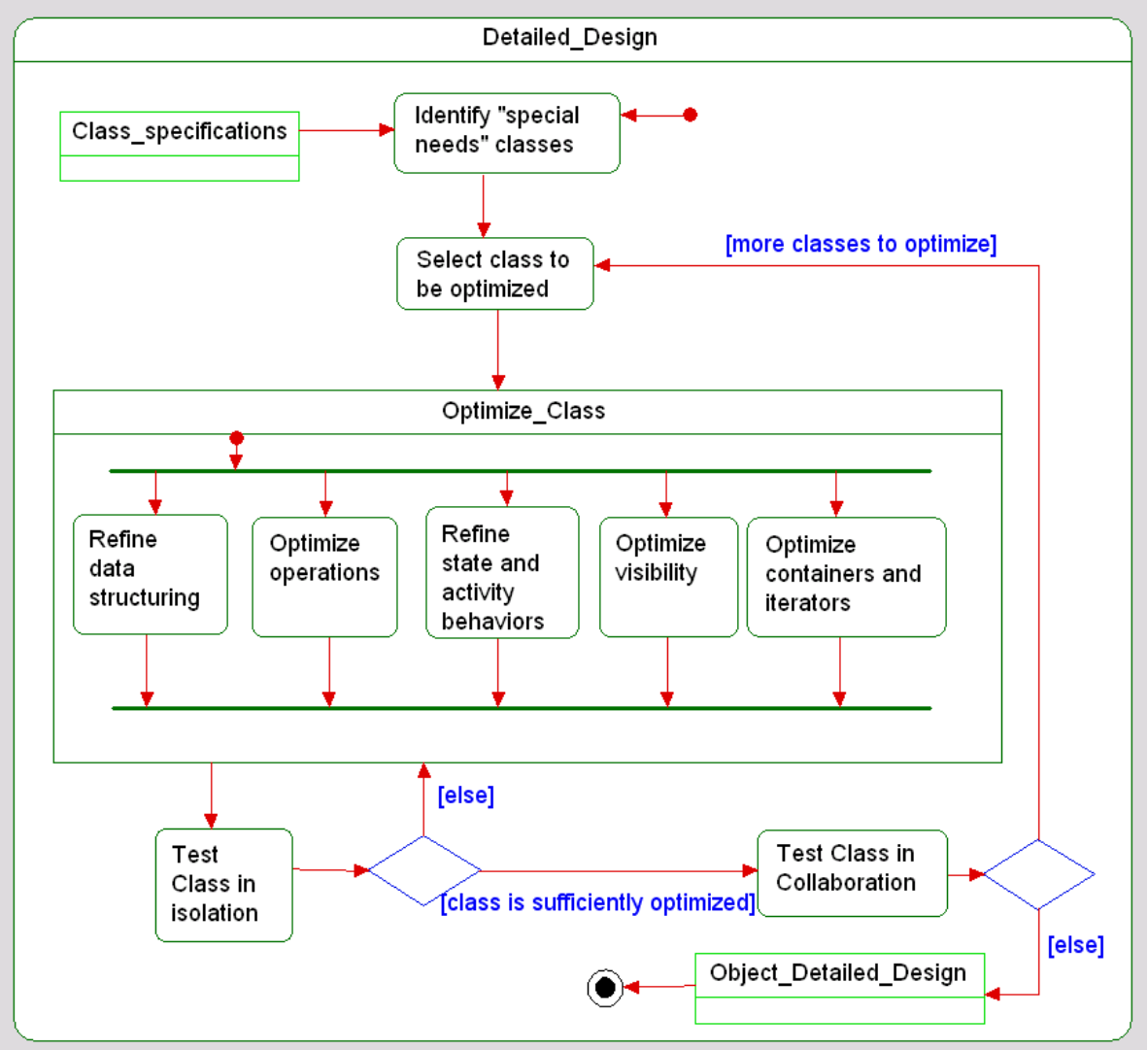

Рис. 2-15: Последовательность работ для детального проектирования

2.2.7 Реализация

Целью фазы реализации является корректное конструирование правильно работающих архитектурных элементов. На этой фазе выполняется генерация программного кода (вне зависимости от того, генерируется ли код автоматически на основе модели, пишется вручную, или используется комбинация эти двух методов), выполняется модульное тестирование программного кода и связанных с ним элементов модели, интеграция с существующим программным кодом, линковка друг с другом отдельных частей архитектурного элемента (возможно, с подключением имеющихся компонентов), а также проверка самого архитектурного элемента на основе модели. Необходимо особо отметить (этот вопрос задается все время), что на фазе реализации *без проблем* может быть подключен уже имеющийся код. На самом деле, было бы *странно*, если бы этого нельзя было сделать.

На фазе реализации создаются следующие артефакты:

- Исходный код, генерируемый на основе элементов модели
- План модульного тестирования, процедуры и результаты (текстовый документ)
- Отчет по ревизии программного кода (текстовый документ)
- Откомпилированные и протестированные программные компоненты

Несложная последовательность работ для данной фазы показана на Рис. 2-16.

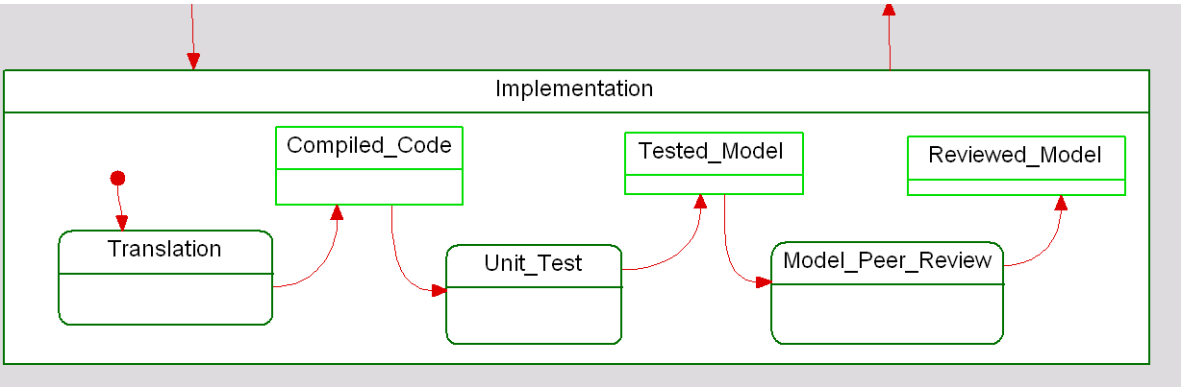

Рис. 2-16: Последовательность работ на фазе реализации

2.2.8 Тестирование

На фазе тестирования прототип собирается из архитектурных элементов и выполняется проверка их совместимости и работоспособности (интеграция), а также проверка что прототип, рассматриваемый как черный ящик, выполняет своё назначение (оценка). Первая из них, интеграция, заключается в сборе интегрированной архитектуры из архитектурных элементов, сконструированных на предыдущей фазе. Тестирование ограничиваются демонстрацией того, что интерфейсы архитектурных элементов используются надлежащим образом, и никакие ограничения не нарушаются. Обычно это производится пошагово в соответствии с планом интеграции, согласно которому архитектурные элементы системы добавляются по одному. Именно на этой фазе для прототипов, имеющих одной из целей программно-аппаратную интеграцию, производится интеграция аппаратных элементов с программными элементами. План и процедуры интеграционного тестирования могут быть разработаны сразу после того, как только была определены подсистемы и компоненты прототипа – то есть, либо в конце стадии системного инжиниринга, либо в конце фазы архитектурного проектирования. На Рис. 2-17 показана последовательность работ по интеграции, выполняемой в конце каждой итерации для объединения отдельных архитектурных элементов, созданных в данной сборке системы.

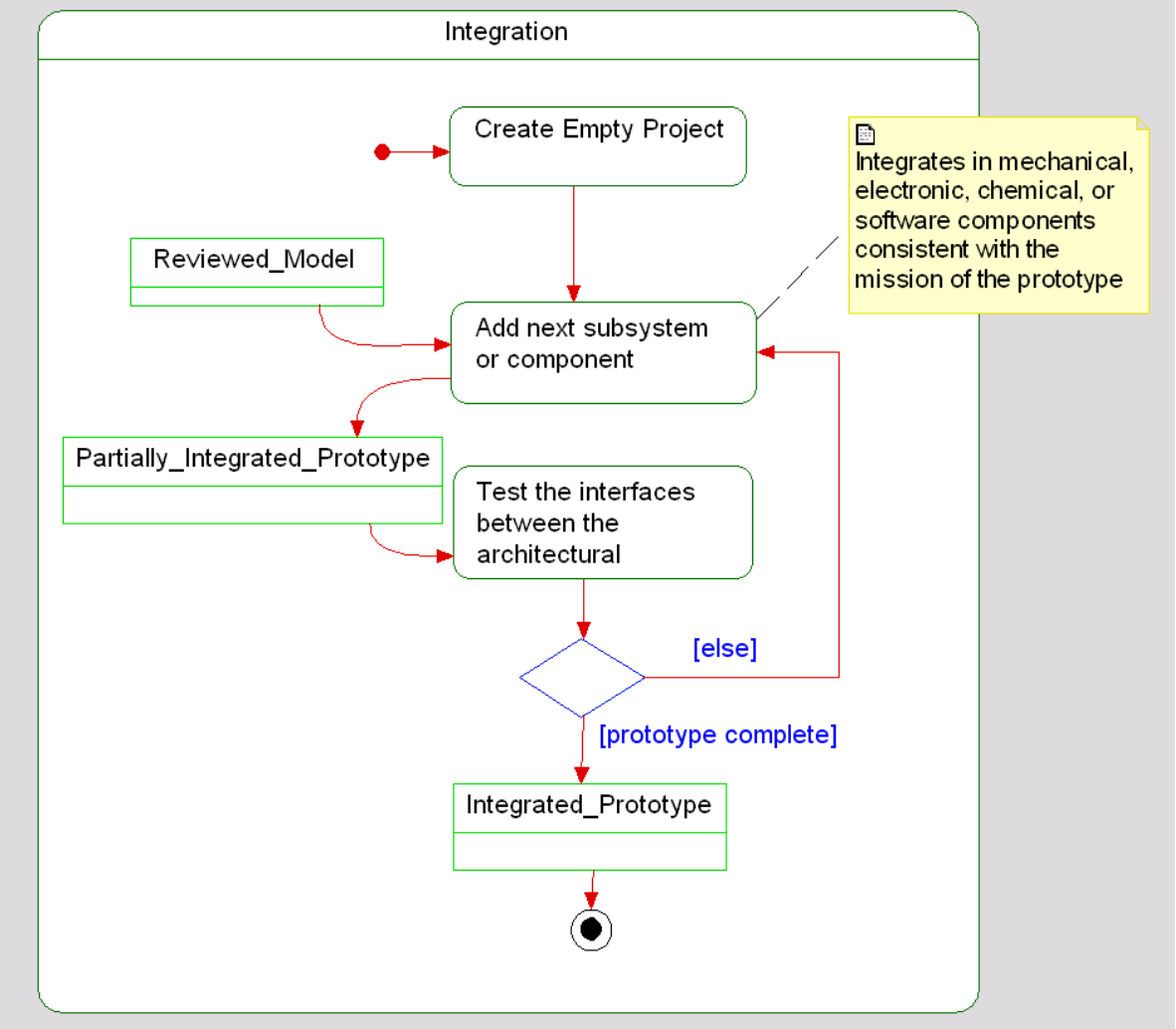

Рис. 2-17: Последовательность работ на фазе интеграции

\
На фазе оценки производится оценка прототипа на соответствие его назначению. Назначение прототипа обычно состоит в реализации небольшого набора вариантов использования и/или снижению небольшого числа рисков. План и процедуры оценки могут быть определены как только для прототипа будут определены и осознаны требования – то есть в конце последовательности работ по определению прототипа в микроцикле.

Если при тестировании обнаруживаются ошибки, они могут быть исправлены немедленно (требуется, если дефект достаточно серьезный) или отложены до следующего прототипа.

Основные артефакты для фазы тестирования:

- План интеграционного тестирования, процедуры и результаты
- План валидации, процедуры и результаты

- Оттестированный, исполняемый прототип
- Отчет о дефектах

Необходимо отметить, что тестовые сценарии для проверки функциональности и качества сервисов создаются в основном на основе диаграмм последовательностей, определенных на фазах анализа (для варианта процесса Harmony- ПО) или системного инжиниринга (для полного процесса Harmony). Для целей регрессионного тестирования используется тестовые сценариев, определенные на предыдущих микроциклах. Вручную добавляются дополнительные тесты для стресс-тестирования, нагрузочного тестирования, тестирование покрытия и тестирование отказоустойчивости. Последовательность работ для фазы оценки показана на Рис. 2-18.

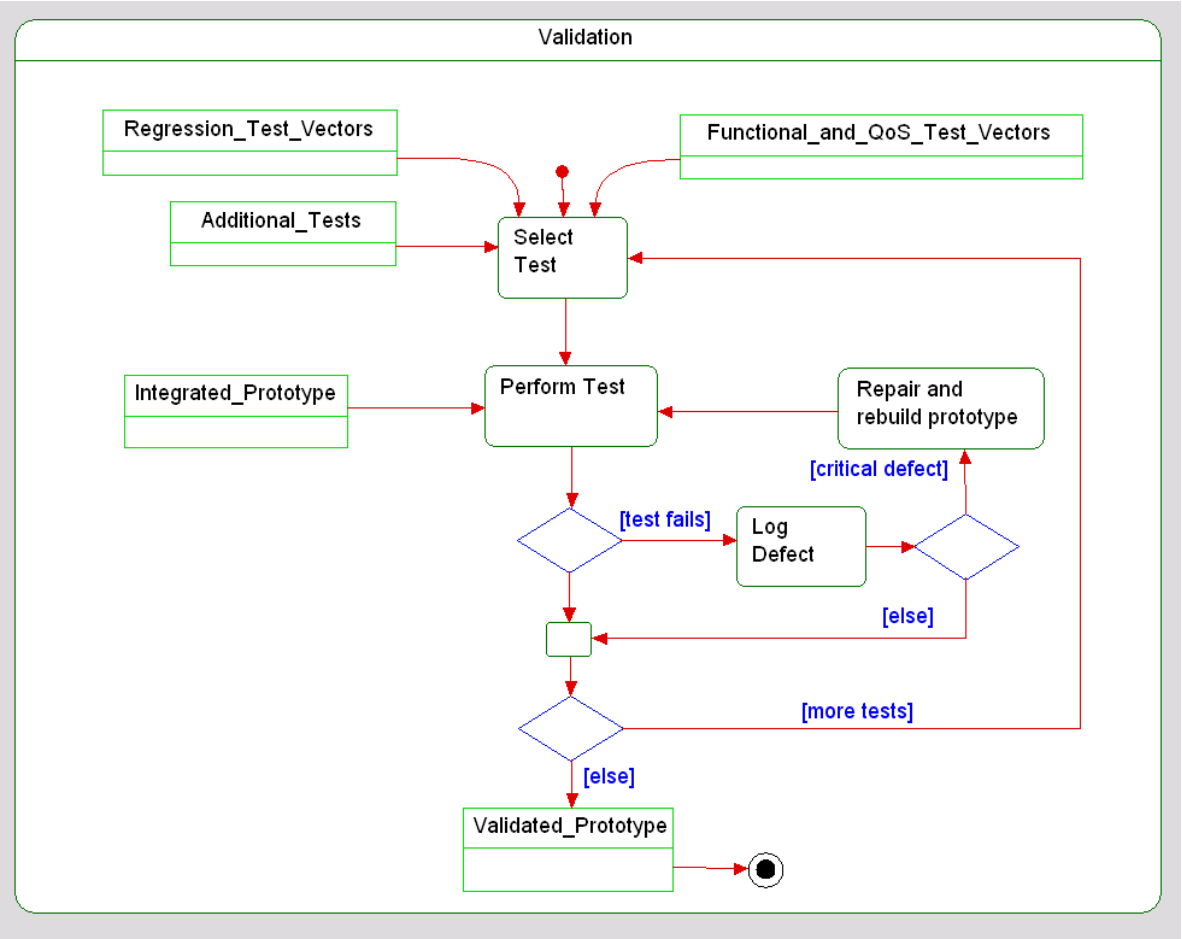

Рис. 2-18: Последовательность работ для валидации

2.2.9 Заключение

В этой главе приведено краткое описание процесса Harmony в обоих его вариантах – полного процесса (включающего в себя системный инжиниринг) и варианта только для программного обеспечения. Процесс Harmony состоит из множества интегрированных последовательностей работ, определенных для каждой фазы

процесса. В его полном варианте спирали по разработке ПО предшествует фаза системного инжиниринга, на которой тщательно прорабатываются все требования к системе и конструируется хорошо согласованная и исполняемая модель вариантов использования. После этого определяется архитектура системы (система разбивается на подсистемы) и требования, в основном представленные в виде операционных контрактов, распределяются по этим подсистемам, для которых они часто групируются в варианты использования уровня подсистем. Сами подсистемы разбиваются на высокоуровневые компоненты, каждый из которых разрабатывается одной инжерной дисциплиной – электронные, механические, химические, программные – и для этих компонентов определяется поведение и логические интерфейсы. После этого созданные модели передаются для дальнейшей разработки командам по разрабоке подсистем, включающим группы разработчиков из различных инженерных дисциплин. В варианте процесса Harmony-ПО возможна небольшая оптимизация по причине отсутствия в нем большого объема работ по совместной разработке программного и аппаратного обеспечения. Вместо того чтобы полностью определять все требования в начале, в Harmony-ПО только выявляются варианты использования, которые не детализируются до момента начала их реализации. Для очередного витка спирали выбирается небольшой набор вариантов использования, которые *при этом* детализируются при помощи диаграмм последовательности, диаграмм состояний, диаграмм деятельности и т.д. Затем создается модель объектного анализа, которая корректно описывает функциональные требования для варианта использования, и выполняется оценка этой модели посредством ее исполнения.

Модель анализа требует дальнейшей оптимизации и введения специфичных технологий реализации, что выполняется в процессе проектирования. Проектирование нацелено на выбор и учет критериев оптимизации, таких как требования к качеству сеовисов. Проектирование производится на трех уровнях абстракции; при архитектурном проектировании оптимизируется вся система в целом, при техническом проектировании оптимизируются кооперации объектов, и при детальном проектировании выполняется оптимизация отдельных объектов.

На выходе фазы реализации создаются высококачественные архитектурные элементы (т.е. подсистемы или компоненты), для которых выполнено модульное тестирование и тщательная ревизия. Эти компоненты при необходимости могут включать имеющийся код и компоненты. На фазе интеграции эти высококачественные компоненты интегрируются друг с другом и с соответствующими аппаратными компонентами, что приводит к созданию интеграционного прототипа. Валидация демонстрирует удовлетворение прототипа его назначению (т.е. требованиям и рискам, которые было необходимо уменьшить). Обнаруженные ошибки устраняются либо на фазе тестирования текущего витка спирали (для критичных дефектор), либо в отведенное на это время на следующем витке.

После окончания текущего витка спирали начинается следующий; при этом выбираются несколько следующих вариантов использования (в варианте процесса Harmony-ПО они также детализируются) и элементов по снижению рисков.

Постепенно итерационный прототип все более функциональным и полным, пока все требования не будут реализованы и оценены. После этого прототип будет выпущен и передан в производство или поставлен заказчику.

В следующих главах мы будем предлагать вам задачи, которые позволят вам приобрести опыт применения данных последовательностей работ при работе над реальными системами. Мы рассмотрим две системы. В приложении А приводится постановка задачи для системы управления светофором. На этом небольшом примере, мы продемонстрируем как описанные последовательности работ применимы к системам среднего масштаба. В приложении В приводится постановки задачи для гораздо более масштабной системы – беспилотный летательный аппарат, содержащий наземные и бортовые подсистемы. Данная более масштабная система позволит нам рассмотреть аспекты масштаба и архитектуры, которые не возникают в небольших системах.

Цели рассмотрения данных систем и предлагаемых далее задач состоит в том, чтобы предоставить вам возможность для получения опыта при *реальной работе*. Нашей целью *не* является создание полных и проверенных моделей для этих систем. В особенности это относится к системе управления воздушным средством. Создание и описание такой системы сделало бы объем данной книги равным нескольким тысяч страниц, много раз больше чем в ней есть сейчас. Тем не менее, масштаб рассматриваемых примеров предоставляет богатый практический материал по использованию подхода к разработке на основе моделей.