Тема 2. **Стандарты и нормативные руководства по системной и программной инженерии.**

Стандарт ISO/IEC 15288 «Системная инженерия - процессы жизненного цикла систем".

ГОСТ 34: Комплекс стандартов на автоматизированные системы.

Ключевые идеи системной инженерии: системный подход, жизненный цикл системы, инжиниринг требований, архитектурный дизайн, процессный подход, проектный подход.

СОДЕРЖАНИЕ ЛЕКЦИИ

2.1. Стандарт ISO 15288 «Системная инженерия - процессы жизненного цикла систем".
2.2. Жизненный цикл системы.
2.3. Представления жизненного цикла системы.
2.4. Жизненный цикл информационной системы
2.5. Модели жизненного цикла
2.6. Выбор модели жизненного цикла

•системного подхода
•жизненного цикла
•инжиниринга требований
•архитектурного дизайна
•процессного подхода
•проектного подхода
•культуры контрактации

2.1. СТАНДАРТ ISO 15288 «СИСТЕМНАЯ ИНЖЕНЕРИЯ - ПРОЦЕССЫ ЖИЗНЕННОГО ЦИКЛА СИСТЕМ".

Системная инженерия применяется для решения проблем, связанных с ростом сложности рукотворных систем. Стандарт ISO 15288[ISO/IEC 15288:2008 Systems and software engineering -- ***System life cycle processes***. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43564], описывающий методы системной инженерии, предписывает иметь описание жизненного цикла системы и его практик. Такое описание требуется для успешного продвижения системы по жизненному циклу. Но стандарт не указывает на методы, с помощью которых требуется создавать подобное описание.

Задачи стандарта:

Дать возможность организациям (внешним и внутренним контракторам) договориться о совмещении замыслов, процессов проектирования, создания, эксплуатации и вывода из эксплуатации самых разных рукотворных систем – от зубочисток до атомных станций, от систем стандартизации до корпораций

Внедрить в практику организации ряд ключевых идей системной инженерии:

- системного подхода
- жизненного цикла
- инжиниринга требований
- архитектурного дизайна
- процессного подхода
- проектного подхода
- культуры контрактации

История создания

Совместная разработка ISO и IEC, активное участие INCOSE
Начало работ в 1996, версии в 2002, 2005 (ГОСТ Р ИСО/МЭК 15288-2005), 2008
Призван гармонизировать так называемое «болото стандартов» системной инженерии (многочисленные стандарты, принятые различными военными ведомствами, государствами, отраслевыми организациями стандартизации)
К разработке стандарта были привлечены специалисты различных областей: системной инженерии, программирования, управления качеством, человеческими ресурсами, безопасностью и пр. Был учтен практический опыт создания систем в правительственных, коммерческих, военных и академических организациях. Стандарт применим для широкого класса систем, но ***его основное предназначение - поддержка создания компьютеризированных систем.***

2.2. ЖИЗНЕННЫЙ ЦИКЛ СИСТЕМЫ

Аббревиатура русск: **ЖЦ**
Аббревиатура англ: **LC (Life Cycle)**
Русский: *«жизненный цикл»*. Английское life cycle в технике ранее означало и переводилось как «срок службы», и иногда даже «срок службы до первого капитального ремонта». «Жизненный цикл» -- это относительно новый перевод. Иногда «цикл» переводят как «период», но такой перевод не устоялся (хотя он и точнее в данном случае: «период жизни» системы). Слово «цикл» не должно смущать – ничего циклического в жизненном цикле нет. Слово «цикл» имеет смысл «типичности», говоря о том, что то же самое происходит и с другими системами.

Формально: жизненный цикл – это смена состояний системы (эволюция системы) в период времени от замысла до прекращения её существования.
Система и жизненный цикл -- близнецы-братья. Мы говорим система -- подразумеваем жизненный цикл, мы говорим жизненный цикл -- подразумеваем система.

Определения.
Определение стандарта ISO/IEC 15288:2008 (Определение: life cycle -- evolution of a system, product, service, project or other human-made entity from conception through retirement (ISO 15288, 4.11):

жизненный цикл **(ЖЦ) – это эволюция системы, продукции, услуги, проекта или иного рукотворного объекта от замысла до прекращения использования.**

Определение стандарта ISO 15704 (Industrial automation systems - Requirements for enterprise-reference architectures and methodologies Системы промышленной автоматизации. Требования к архитектуре эталонных предприятий и методологии. Описывает эталонную архитектуру предприятия и средства реализации проектов в рамках полнрго жизненного цикла предприятия):

жизненный цикл **(ЖЦ) – это конечный набор основных фаз и шагов, которые система проходит на протяжении всей истории существования.**

Каждая система, вне зависимости от ее вида и масштаба, проходит весь свой жизненный цикл согласно некоторому описанию. Продвижение системы по частям этого описания и есть жизненный цикл системы. Описание жизненного цикла, таким образом, — *это концептуальная сегментация по стадиям*, способствующим планированию, разворачиванию, эксплуатации и поддержке целевой системы.
Стадии (табл. 2.1) представляют наиболее крупные периоды жизненного цикла, ассоциируемые с системой, и соотносятся с состояниями описания системы или реализацией системы как набора продуктов или услуг. Стадии описывают основные контрольные точки продвижения и успехов системы по ходу жизненного цикла. Такие сегменты дают упорядоченное продвижение системы через установленные пересмотры выделения ресурсов, что снижает риски и обеспечивает удовлетворительное продвижение. Основной причиной применения описаний жизненного цикла является потребность в принятии решений по определенным критериям до продвижения системы на следующую стадию.

1	**Формирование концепции**	**Анализ потребностей, выбор концепции и проектных решений**
2	Разработка	Проектирование системы
3	Реализация	Изготовление системы
4	Эксплуатация	Ввод в эксплуатацию и использование системы
5	Поддержка	Обеспечение функционирования системы
6	Снятие с эксплуатации	Прекращение использования, демонтаж, архивирование системы

Комментарий: жизненный цикл – всегда жизненный цикл конкретной системы. Не бывает «жизненного цикла» кроме как в текстах стандартов, в жизни всегда «жизненный цикл X», где X – название целевой системы. Процессы жизненного цикла – это те процессы, которые акторы выполняют над/с системой, и которые меняют состояние системы, заставляя ее эволюционировать в ходе её жизненного цикла. «Управление жизненным циклом» -- общепринятое название подхода к описанию процессов жизненного цикла (а часто и название самой группы процессов жизненного цикла, описанных с использованием такого подхода).

2.3. Представления жизненного цикла системы

У системы есть два основных представления: целевое (архитектурное, чаще всего структурное в своей основе, плюс процессы времени эксплуатации системы) и жизненного цикла (развертка во времени жизненного цикла - процессы обеспечивающих систем). Можно обсуждать, насколько каждое из этих представлений является частью другого, но для надлежащего описания системы всегда нужно использовать какое-то представление жизненного цикла.

Прежде всего, нужно различить жизненный цикл (иногда, ограничиваясь только инженерией, но не полным ЖЦ говорят также delivery process, изредка для софта -- software process) и другие "процессные представления" -- трансакции DEMO, логические "бизнес-процессы" (практики), workflows, проектные представления (подробнее -- http://ailev.livejournal.com/904643.html).

Хотя есть множество подходов, при которых все эти разные аспекты описаний организации и методов ее работы смешиваются.

Модель жизненного цикла отражает различные состояния системы, начиная с момента возникновения необходимости в данной ИС и заканчивая моментом ее полного выхода из употребления.

Модель жизненного цикла - структура, содержащая процессы, действия и задачи, которые осуществляются в ходе разработки, функционирования и сопровождения программного продукта в течение всей жизни системы, от определения требований до завершения ее использования.

Языков представления жизненного цикла и текстовых и графических нотаций для этих языков много, ограничимся для примера лишь следующими:

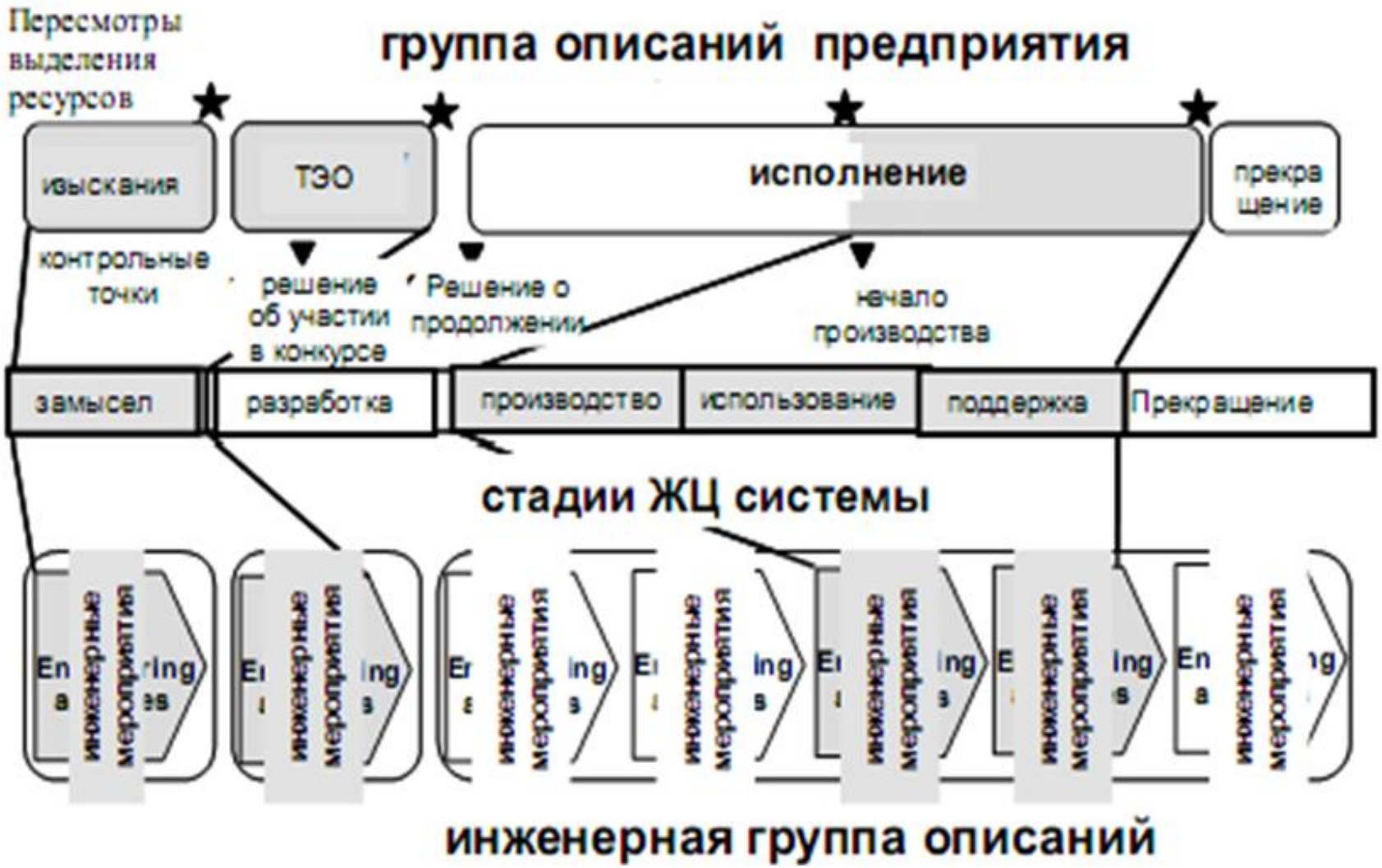
«Нарезанная колбаска»

V-диаграмма

Нарезанная колбаска".

Просто перечисление стадий жизненного цикла их названиями, для выразительности названия упакованы в отрезки "колбаски"

Замысел	Разработка	Производство	Использование	Поддержка	Прекращение использования



Вокруг традиционной «колбаски» могут указываться еще две дополнительных: как ЖЦ видят менеджеры (лица, управляющие проектом), и как ЖЦ видят инженеры(лица, реализующие проект)

Жизненные циклы наблюдаются в историях отдельных товаров и потребностей, торговых марок, предприятий, целых индустрий и рынков. Жизненный цикл неотделим от конкретной системы, поэтому особенности разных систем порождают большое разнообразие экземпляров «колбасок» жизненных циклов

V-диаграмма. Самое знаменитое представление жизненного цикла

Одним из самых известных представлений жизненного цикла системы является V-диаграмма, задающая логику системной инженерии.

Что дает V-диаграмма:
Разработка продукта происходит в полном соответствии с требованиями.
Учет изменений на всех уровнях разработки
Тесты, проверки, сертификация проверят требования
Обеспечивается сквозной мониторинг производства продукта

2.4. Жизненный цикл информационной системы

Совокупность стадий и этапов, которые проходит ИС в своем развитии от момента принятия решения о создании системы до момента прекращения функционирования системы, называется жизненным циклом ИС.

Содержание жизненного цикла разработки ИС сводится к выполнению следующих стадий:

1. Планирование и анализ требований (предпроектная стадия) – системный анализ. Проводится исследование и анализ существующей информационной системы, определяются требования к создаваемой ИС, формируются технико-экономическое обоснование (ТЭО) и техническое задание (ТЗ) на разработку ИС;

2. Проектирование (техническое и логическое проектирование). В соответствии с требованиями формируются состав автоматизируемых функций (функциональная архитектура) и состав обеспечивающих подсистем (системная архитектура), проводится оформление технического проекта ИС;
3. Реализация (рабочее и физическое проектирование, кодирование). Разработка и настройка программ, формирование и наполнение баз данных, формулировка рабочих инструкций для персонала, оформление рабочего проекта;
4. Внедрение (опытная эксплуатация). Комплексная отладка подсистем ИС, обучение персонала, поэтапное внедрение ИС в эксплуатацию по подразделениям организации, оформление акта о приемо-сдаточных испытаниях ИС;
5. Эксплуатация ИС (сопровождение, модернизация). Сбор рекламаций и статистики о функционировании ИС, исправление недоработок и ошибок, оформление требований к модернизации ИС и ее выполнение (повторение стадий 2-5).

основное содержание стадий и этапов жизненного цикла ИС

Системный анализ. Основными целями этапа являются:
* формулировка потребностей в новой ИС (определение всех недостатков существующей ИС);
* выбор направления и определение экономической обоснованности проектирования ИС.

Системный анализ ИС начинается с описания и анализа функционирования рассматриваемого объекта в соответствии с требованиями (целями), которые предъявляются к нему. В результате этого этапа выявляются недостатки существующей ИС, на основе которых формулируется потребность в совершенствовании системы управления этим объектом, и ставится задача определения экономически обоснованной необходимости автоматизации определенных функций управления (создается технико-экономическое обоснование проекта ИС). После определения этой потребности возникает проблема выбора направлений совершенствования объекта на основе выбора программно-технических средств. Результаты оформляются в виде технического задания на проект, в котором отражаются технические условия и требования к ИС, а также ограничения на ресурсы проектирования. Требования к ИС определяются в терминах функций, реализуемых системой.

Этап проектирования предполагает:
* проектирование функциональной архитектуры ИС, которая отражает структуру выполняемых функций;
* проектирование системной архитектуры ИС (состав обеспечивающих подсистем);
* реализацию проекта.

Формирование функциональной архитектуры, которая представляет собой совокупность функциональных подсистем и связей между ними, является наиболее ответственным и важным этапом с точки зрения качества всей последующей разработки ИС.

Построение системной архитектуры на основе функциональной предполагает определение элементов и модулей информационного, технического, программного обеспечения и других обеспечивающих подсистем, связей по информации и управлению между выделенными элементами и разработку технологии обработки информации.

Реализация включает разработку программ и инструкций для пользователей, создание информационного обеспечения, включая наполнение баз данных. **Внедрение** разработанного проекта разделяется на опытное и промышленное.
Этап опытного внедрения подразумевает проверку работоспособности элементов и модулей проекта, устранение ошибок на уровне элементов и связей между ними. **Этап сдачи в промышленную эксплуатацию** заключается в организации проверки проекта на уровне функций, контроля соответствия его требованиям, сформулированным на стадии системного анализа.

Важной особенностью жизненного цикла ИС является его повторяемость (цикличность) "системный анализ — разработка — сопровождение — системный анализ". Это соответствует представлению об ИС как о развивающейся, динамической системе.

При первом выполнении стадии "Разработка" создается проект ИС, а при последующих реализациях данной стадии осуществляется модификация проекта для поддержания его в актуальном состоянии.

2.5 МОДЕЛИ ЖИЗНЕННОГО ЦИКЛА ИС

С точки зрения реализации перечисленных выше аспектов в технологиях проектирования ИС модели жизненного цикла, определяющие порядок выполнения стадий и этапов, претерпевали существенные изменения. Среди известных моделей жизненного цикла можно выделить следующие:

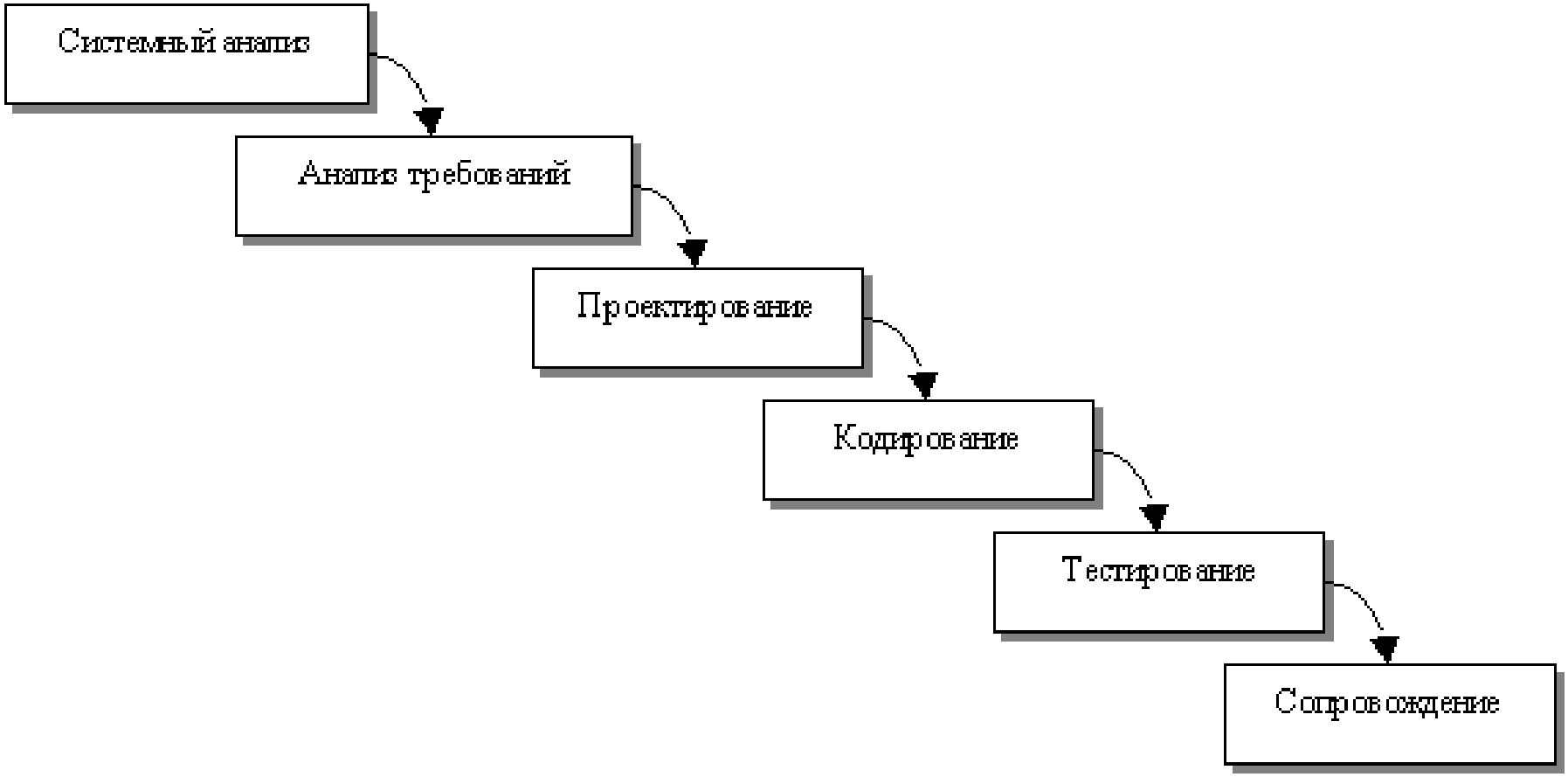
- каскадная модель (до 70-х годов) — последовательный переход на следующий этап после завершения предыдущего;
- итерационная модель (70-80-е годы) — с итерационными возвратами на предыдущие этапы после выполнения очередного этапа;
- спиральная модель (80-90-е годы) — прототипная модель, предполагающая постепенное расширение прототипа ИС.

КАСКАДНАЯ МОДЕЛЬ

В ***каскадной модели*** переход на следующий, иерархически нижний этап происходит только после полного завершения работ на текущем этапе. Достоинство каскадной модели заключается в планировании времени осуществления всех этапов проекта, упорядочении хода конструирования.

Недостатки каскадной модели:

- реальные проекты часто требуют отклонения от стандартной последовательности шагов (недостаточно гибкая модель);
- цикл основан на точной формулировке исходных требований к ПО (реально в начале проекта требования заказчика определены лишь частично);
- результаты проекта доступны заказчику только в конце работы.


Системный анализ
Анализ требований
Проектирование
Кодирование
Тестирование
Сопровождение

Итерационная модель. Построение комплексных ИС подразумевает согласование проектных решений, получаемых при реализации отдельных задач. Подход к проектированию "снизу вверх" предполагает необходимость таких итерационных возвратов, когда проектные решения по отдельным задачам объединяются в общие системные решения, и при этом возникает потребность в пересмотре ранее сформулированных требований. Вследствие большого числа итераций возникают рассогласования и несоответствия в выполненных проектных решениях и документации.

Модель прототипирования жизненного цикла разработки ПО

Ставшее классикой произведение Фреда Брукса (Fred Brook) под названием "Легендарный человек-месяц" (The Mythical Man-Month) сегодня столь же актуально, как и в 1975 году. Технологии радикально изменили мир, но многие недостатки менеджмента программных проектов по-прежнему те же. Десятки лет тому назад Брукс сказал:

"В большинстве проектов первая построенная система едва ли пригодна к употреблению. Она может быть слишком медленной, слишком объемной, неудобной в использовании или обладать всеми тремя перечисленными недостатками. Нет другого выбора, кроме как начать с самого начала, приложив все усилия, и построить модернизированную версию, в которой решались бы все три проблемы...

В случае, когда в проекте используется новая системная концепция или новая технология, разработчик вынужден построить систему, которой впоследствии не воспользуется, поскольку даже при наилучшем планировании невозможно предвидеть достижение нужного результата.
Следовательно, вопрос менеджмента заключается не в том, создавать или нет экспериментальную систему, которой затем не воспользуются. Вы в любом случае так и сделаете. Единственный вопрос в том, нужно ли планировать создание продукта одноразового использования заранее или обещать поставить его заказчикам..."

Именно эта концепция построения экспериментальной, или прототипной системы привела к возникновению "структурной", "эволюционной" модели быстрого прототипирования (RAD), и спиральной модели. В своей боле поздней, в равной степени полной плодотворных идей работе под названием "No Silver Bullet, the Essence and Accidents of Programming" Брукс считает, что большинство ошибок, возникающих при разработке ПО, все же связаны с неправильным пониманием концепции системы, а не с синтаксисом или логикой. Разработка ПО всегда будет трудной задачей, и мы никогда не найдем чудодейственную панацею или "серебряную пулю". Он подчеркивает положительный момент в применении методов быстрого прототипирования:

"Самой тяжелой составляющей процесса построения программной системы является принятие однозначного решения о том, что именно необходимо построить. Ни одна из остальных составляющих работы над концепцией не представляет собой такую трудность, как определение детальных технических требований, включая все аспекты соприкосновения продукта с людьми, машинами и другими программными системами. Ни одна другая составляющая работы в такой степени не наносит ущерб полученной в результате системе, если она выполнена неправильно. Именно эту составляющую процесса разработки тяжелее всего исправить на более поздних этапах.

Следовательно, самая важная функция, которую выполняет разработчик клиентских программ, заключается в итеративном извлечении и уточнении требований к продукту. Ведь на самом деле клиент не имеет представления о том, что именно он хочет получить.

На данный момент времени одной из самых обещающих среди технологических попыток, сосредоточенных на сущности, а не на трудностях решения проблемы разработки ПО, является разработка методов и средств для ускоренного прототипиро-вания систем как составляющей итеративной спецификации требований".

Уотте Хэмфри (Watts Humphrey), который известен как вдохновитель создания модели CMM, разработанной Институтом SEI, поддерживает Брукса в его подходе к важности требований и их разработки:
"В большинстве систем заключен основной принцип, который включает в себя больше, чем незначительное эволюционное изменение. Система может изменить само эксплуатационное окружение. Поскольку пользователи могут рассуждать о том или ином явлении только в рамках известного им окружения, требования к таким системам всегда формулируются в рамках текущего окружения. Следовательно, эти требования непременно будут неполными, неточными и обманчивыми.

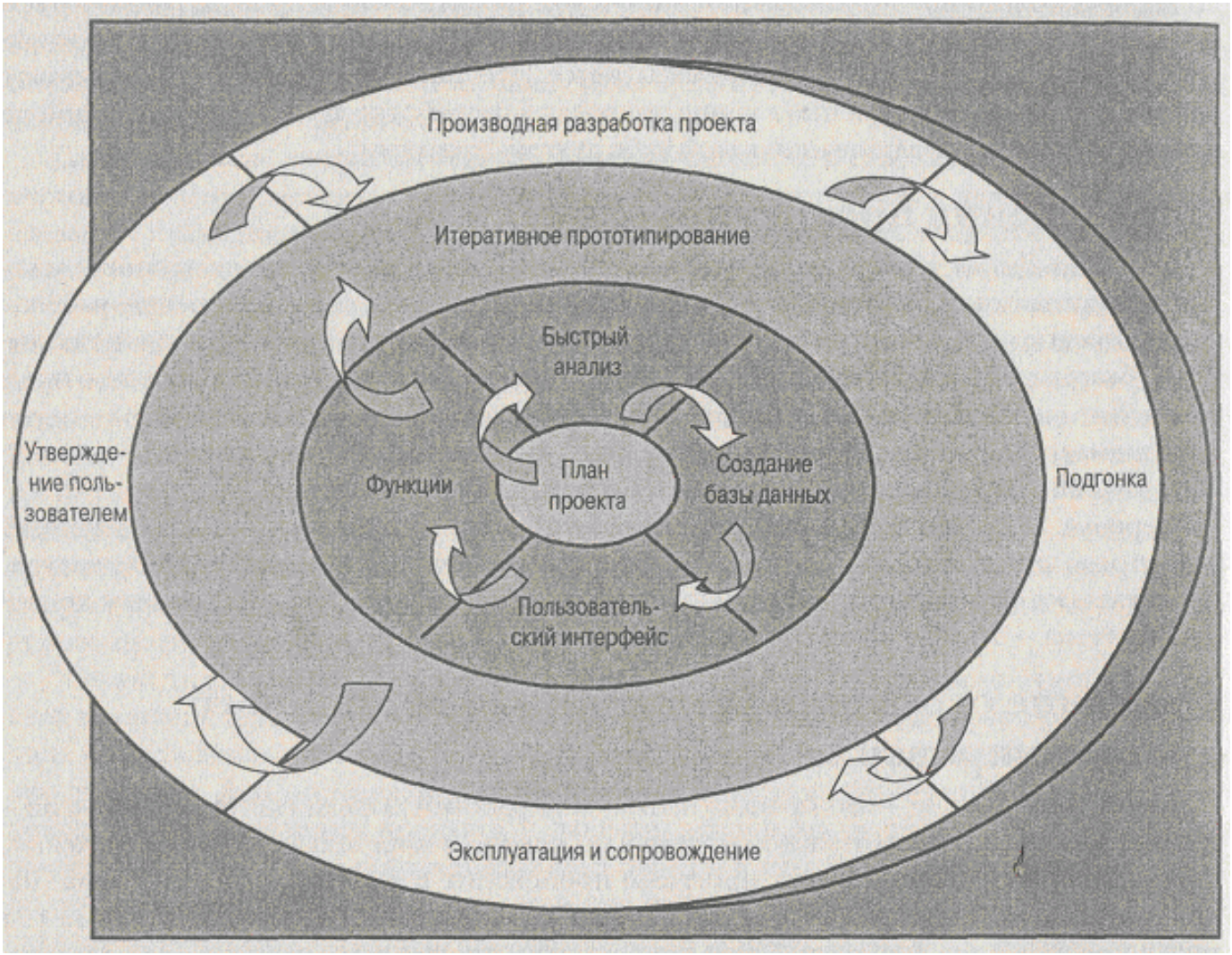
Главной задачей для системного разработчика является изобретение процесса разработки, с помощью которого можно будет обнаружить, определить и разработать реальные требования. Этого можно достичь только при максимальном включении пользователя в процесс разработки и зачастую с помощью периодического тестирования прототипов или версий, полученных на ранних этапах разработки. Оказывается, что такие процессы всегда занимают больше времени, но неизменно в конце приводят к разработке лучшей системы намного быстрее, чем при использовании какой-либо другой стратегии".

Определения прототипирования

Согласно Джону Коннэллу (Connell) и Линде Шафер (Shafer), эволюционным ускоренным прототипом является "легко поддающаяся модификации и расширению рабочая модель предполагаемой системы, не обязательно представляющая собой все свойства системы, благодаря которой пользователи данного приложения получают физическое представление о ключевых частях системы до ее непосредственной реализации; это — легко создаваемая, без труда поддающаяся модификации, максимально расширяемая, частично заданная рабочая модель основных аспектов предполагаемой системы" .
Бернард Боар (Bernard Boar) определил прототип как "метод, предназначенный для определения требований, при котором потребности пользователя извлекаются, представляются и разрабатываются посредством построения рабочей модели конечной системы — быстро и в требуемом контексте".

Описание структурной модели эволюционного прототипирования

Прототипирование — это процесс построения рабочей модели системы. Прототип — это эквивалент экспериментальной модели или "макета" в мире аппаратного обеспечения.
Выполнение эволюционных программ происходит в рамках контекста плана, направленного на достижение предельно высокой производительности. Этот метод также предполагает, что разработка инкрементов программы очевидна для пользователя, который принимает участие в течение всего процесса разработки.



Структурная эволюционная модель быстрого прототипирования

"Быстрая" частичная реализация системы создается перед этапом определения требований или на его протяжении. Конечные пользователи системы используют ускоренный прототип, а затем путем обратной связи сообщают о своем достижении команде, работающей над проектом, для дальнейшего уточнения требований к системе.

Процесс уточнения продолжается до тех пор, пока пользователь не получит то, что ему требуется. После завершения процесса определения требований путем разработки ускоренных прототипов, получают детальный проект системы, а ускоренный прототип регулируется при использовании кода или внешних утилит, в результате чего получают конечный рабочий продукт.

В идеале можно вывести, при чем без излишних затрат, модель прототипирования высокого качества, не экономя на документации, анализе, проектировании, тестировании и т.д. Следовательно, она получила название "структурной модели быстрого прототипирования", как показано на рис

Начало жизненного цикла разработки помещено в центре эллипса. Пользователь и программист разрабатывают предварительный план проекта, руководствуясь при этом предварительными требованиями. Используя методы ускоренного анализа, пользователь и программист совместно работают над определением требований и спецификаций для важнейших частей воображаемой системы. Планирование проекта — это первое действие на этапе быстрого анализа, с помощью которого получают документ, описывающий в общих чертах примерные графики и результативные данные.

Таким образом, создается план проекта, а затем выполняется быстрый анализ, после чего проектируется база данных, пользовательский интерфейс и разработка функций. Второе действие — это быстрый анализ, на протяжении которого предварительные опросы пользователей используются для разработки умышленно неполной высокоуровневой модели системы на уровне документации. В результате выполнения этой задачи получают документ, содержащий частичную спецификацию требований, который используется для построения исходного прототипа, создаваемого на последующих трех этапах. Дизайнер конструирует модель (используя для этого инструментальные средства), то есть частичное представление системы, которое включает в себя только те базовые свойства, которые необходимы для удовлетворения требований заказчика. Затем начинается итерационный цикл быстрого прототипирования. Разработчик проекта демонстрирует прототип, а пользователь оценивает его функционирование.

После этого определяются проблемы, над устранением которых совместно работают пользователь и дизайнер. Этот процесс продолжается до тех пор, пока пользователь не будет удовлетворен тем, каким образом система отображает поставленные к ней требования. Команда разработчиков проекта продолжает выполнять этот процесс до тех пор, пока пользователь не согласится, что быстрый прототип в точности отображает системные требования. Создание базы данных представляет собой первую из этих двух фаз. После создания исходной базы данных можно начать разработку меню, после чего следует разработка функций, то есть создается рабочая модель. Затем модель демонстрируют пользователю с целью получения предложений по ее усовершенствованию, которые объединяются в последовательные итерации до тех пор, пока рабочая модель не окажется удовлетворительной. Затем получают официальное одобрение пользователем функциональных возможностей прототипа. После этого создается документ предварительного проекта системы.

Основным компонентом является фаза итерации прототипа, на протяжении которого при использовании сценариев, предоставленных рабочей моделью, пользователь может разыграть роли и потребовать, чтобы последовательное уточнение модели продолжалось до тех пор, пока не будут удовлетворены все функциональных требования. Получив одобрение пользователя, быстрый прототип преобразуют детальный проект, и систему настраивают на производственное использование. Именно на этом этапе настройки ускоренный прототип становится полностью действующей системой, которая заменяет собой частичную систему, полученную в итерационном цикле прототипирования.

Детализированный проект можно также получить на основе прототипов. В этом случае настройка прототипа выполняется при использовании кода или внешних утилит. Дизайнер использует утвержденные требования в качестве основы для проектирования производственного ПО.

При разработке производственной версии программы, может возникнуть необходимость в дополнительной работе. Может понадобиться более высокий уровень функциональных возможностей, различные системные ресурсы, необходимых для обеспечения полной рабочей нагрузки, или ограничения во времени. После этого следуют тестирование в предельных режимах, определение измерительных критериев и настройка, а затем, как обычно, функциональное сопровождение. Заключительная фаза представляет собой функционирование и сопровождение, отображают действия, направленные на перемещение системы в стадию производственного процесса. Не существует "правильного" способа использования метода прототипирования. Полученный результат может не пригодиться, может быть использован в качестве основания для последующей модернизации или превращен в продукт, используемого процесса и желаемого качества в зависимости от исходных целей. Понятно, что при использовании *эволюционного* прототипирования снижаются затраты и оптимизируется соблюдение графиков, поскольку каждый из его компонентов находит свое применение.

Преимущества структурной эволюционной модели быстрого прототипирования

При использовании структурной эволюционной модели быстрого прототипирования для приемлемого проекта проявляются следующие преимущества:

- конечный пользователь может "увидеть" системные требования в процессе их сбора командой разработчиков; таким образом, взаимодействие заказчика с системой начинается на раннем этапе разработки;
- исходя из реакции заказчиков на демонстрации разрабатываемого продукта, разработчики получают сведения об одном или нескольких аспектах поведения системы, благодаря чему сводится к минимуму количество неточностей в требованиях;
- снижается возможность возникновения путаницы, искажения информации или недоразумений при определении системных требований, что несомненно приводит к созданию более качественного конечного продукта;

- в процесс разработки можно внести новые или неожиданные требования пользователя, что порой необходимо, так как реальность может отличаться от концептуальной модели реальности;
- модель представляет собой формальную спецификацию, воплощенную в рабочую модель;
- модель позволяет выполнять гибкое проектирование и разработку, включая несколько итераций на всех фазах жизненного цикла;
- при использовании модели образуются постоянные, видимые признаки прогресса в выполнении проекта, благодаря чему заказчики чувствуют себя уверенно;
- возможность возникновения разногласий при общении заказчиков с разработчиками минимизирована;
- ожидаемое качество продукта определяется при активном участии пользователя в процесс на ранних фазах разработки;

- возможность наблюдать ту или иную функцию в действии пробуждает очевидную необходимость в разработке функциональных дополнительных возможностей;
- благодаря меньшему объему доработок уменьшаются затраты на разработку;
- благодаря тому что проблема выявляется до привлечения дополнительных ресурсов сокращаются общие затраты;
- обеспечивается управление рисками;
- документация сконцентрирована на конечном продукте, а не на его разработке;
- принимая участие в процессе разработки на протяжении всего жизненного цикла, пользователи в большей степени будут довольны полученными результатами.

Недостатки структурной эволюционной модели быстрого прототипирования:

- модель может быть отклонена из-за создавшейся среди консерваторов репутации о ней как о "разработанном на скорую руку" методе;
- разработанные "на скорую руку" прототипы, в отличие от эволюционных ускоренных прототипов, страдают от неадекватной или недостающей документации;
- если цели прототипирования не согласованы заранее, процесс может превратиться в упражнение по созданию хакерского кода;

- с учетом создания рабочего прототипа, качеству всего ПО или долгосрочной эксплуатационной надежности может быть уделено недостаточно внимания.
- иногда в результате использования модели получают систему с низкой рабочей характеристикой, особенно если в процессе ее выполнения пропускается этап подгонки;
- при использовании модели решение трудных проблем может отодвигаться на будущее. В результате это приводит к тому, что последующие полученные продукты могут не оправдать надежды, которые возлагались на прототип;
- если пользователи не могут участвовать в проекте на итерационной фазе быстрого прототипирования жизненного цикла, на конечном продукте могут отразиться неблагоприятные воздействия, включая проблемы, связанные с его качественной характеристикой;
- на итерационном этапе прототипирования быстрый прототип представляет собой частичную систему. Если выполнение проекта завершается досрочно, у конечного пользователя останется только лишь частичная система;
- несовпадение представлений заказчика и разработчиков об использовании прототипа может привести к созданию другого пользовательского интерфейса;

- заказчик может предпочесть получить прототип, вместо того, чтобы ждать появления полной, хорошо продуманной версии;
- если язык или среда прототипирования не сочетаются с производственным языком или окружением, всесторонняя реализация продукционной системы может быть задержана;
- прототипирование вызывает зависимость и может продолжаться слишком долго. Нетренированные разработчики могут попасть в так называемый цикл "кодирование — устранение ошибок" (code-and-fix cycle), что приводит к дорогостоящим незапланированным итерациям прототипирования;
- разработчики и пользователи не всегда понимают, что когда прототип превращается в конечный продукт, все еще существует необходимость в традиционной Документации. Если она отсутствует, модифицировать модель на более поздних этапах может оказаться более дорогостоящим занятием, чем просто не воспользоваться созданным прототипом;
- когда заказчики, удовлетворенные прототипом, требуют его немедленной поставки, перед менеджером программного проекта возникает соблазн пойти им навстречу;

- на заказчиков могут неблагоприятно повлиять сведения об отличии между прототипом и полностью разработанной системой, готовой к реализации;
- на заказчиков может оказать негативное влияние тот факт, что они не располагают информацией о точном количестве итераций, которые будут необходимы;
- на разработку системы может быть потрачено слишком много времени, так как итерационный процесс демонстрации прототипа и его пересмотр могут продолжаться бесконечно без надлежащего управления процессом. У пользователей может возникнуть стремление пополнять список элементов, предназначенных для прототипирования, до тех пор, пока проект ни достигнет масштаба, значительно превышающего рамки, определенные анализом осуществимости проектного решения;
- при выборе инструментальных средств прототипирования (операционные системы, языки и малопродуктивные алгоритмы) разработчики могут остановить свой выбор на менее подходящем решении, только чтобы продемонстрировать свои способности;
- структурные методы не используются, чтобы не помешать выполнению анализа. При прототипировании необходимо провести "реальный" анализ требований, осуществить проектирование и обратить внимание на качество с целью создания программы, допускающей сопровождение, точно так же, как и в любой другой модели жизненного цикла (хотя на эти действия может понадобиться меньше времени и ресурсов).

Область применения структурной эволюционной модели быстрого прототипирования

Менеджер проекта может быть уверен в необходимости применения структурной эволюционной модели быстрого прототипирования, если:

- требования не известны заранее;
- требования не постоянны или могут быть неверно истолкованы или неудачно сформулированы;
- следует уточнить требования;
- существует потребность в разработке пользовательских интерфейсов;
- нужна проверка концепции;
- осуществляются временные демонстрации;
- построенное по принципу структурной модели, эволюционное быстрое прототипирование можно успешно использовать в больших системах, в которых некоторые модели подвергаются прототипированию, а некоторые— разрабатываются более традиционным образом;

- выполняется новая, не имеющая аналогов разработка (в отличие от эксплуатации продукта на уже существующей системе);
- требуется уменьшить неточности в определении требований; т.е. уменьшается риск создания системы, которая не имеет никакой ценности для заказчика;
- требования подвержены быстрым изменениям, когда заказчик неохотно соглашается на фиксированный набор требований или если о прикладной программе отсутствует четкое представление;
- разработчики не уверены в том, какую оптимальную архитектуру или алгоритмы следует применять;
- алгоритмы или системные интерфейсы усложнены;
- требуется продемонстрировать техническую осуществимость, когда технический риск высок;
- задействованы высокотехнологические системы с интенсивным применением ПО, где можно лишь обобщенно, но не точно сформулировать требования, лежащие за пределами главной характеристики;
- разрабатывается ПО, особенно в случае программ, когда проявляется средняя и высокая степень риска;

- осуществляется применение в комбинации с каскадной моделью: на начальном этапе проекта используется прототипирование, а на последнем — фазы каскадной модели с целью обеспечения функциональной эффективности системы и качества;
- прототипирование всегда следует использовать вместе с элементами анализа и проектирования, применяемыми при объектно-ориентированной разработке. Быстрое прототипирование особенно хорошо подходит для разработки интенсивно используемых систем пользовательского интерфейса, таких как индикаторные панели для контрольных приборов, интерактивные системы, новые в своем роде продукты, а также системы обеспечения принятия решений, среди которых можно назвать подачу команд, управление или медицинскую диагностику.

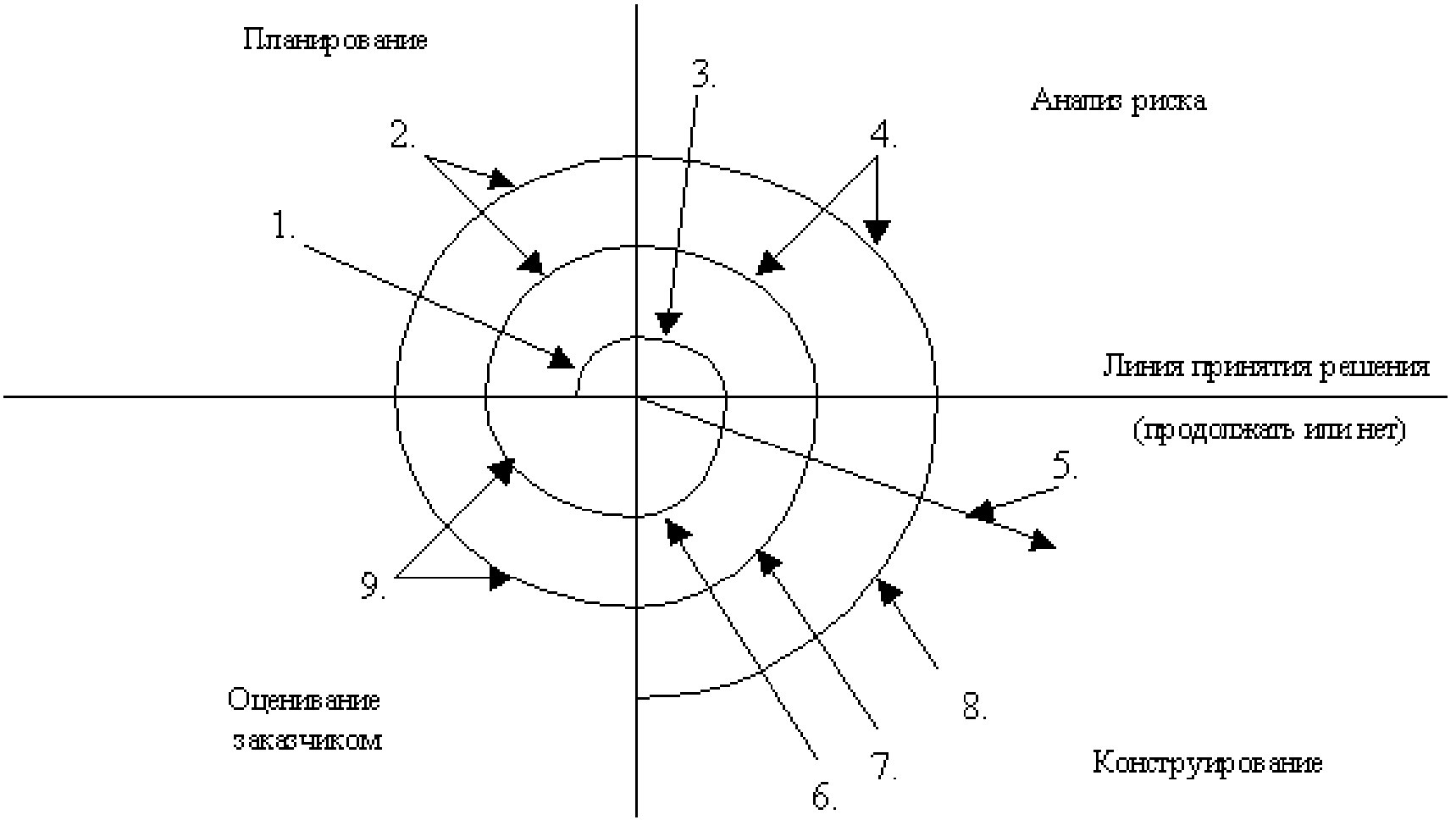
Спиральная модель

Спиральная модель(рис.2) — классический пример применения эволюционной стратегии конструирования.

Где, 1. начальный сбор требований и планирование проекта; 2. та же работа, но на основе рекомендаций заказчика; 3. анализ риска на основе начальных требований; 4. анализ риска на основе реакции заказчика; 5. переход к комплексной системе; 6. начальный макет системы; 7. следующий уровень макета; 8. сконструированная система; 9. оценивание заказчиком.

Как показано на рис. , спиральная модель определяет четыре действия, представляемые четырьмя квадрантами спирали:

- планирование — определение целей, вариантов и ограничений;
- анализ риска — анализ вариантов и распознавание (выбор) риска;
- конструирование — разработка продукта следующего уровня;
- оценивание — оценка заказчиком текущих результатов конструирования.



Спиральная модель

Интегрирующий аспект спиральной модели очевиден при учете радиального измерения спирали. С каждой итерацией по спирали (продвижением от центра к периферии) строятся все более полные версии ПО.

Спиральная модель жизненного цикла ИС реально отображает разработку программного обеспечения; позволяет явно учитывать риск на каждом витке эволюции разработки; включает шаг системного подхода в итерационную структуру разработки; использует моделирование для уменьшения риска и совершенствования программного изделия.

Недостатками спиральной модели являются:

- новизна (отсутствует достаточная статистика эффективности модели);
- повышенные требования к заказчику;

МОДЕЛЬ БЫСТРОЙ РАЗРАБОТКИ ПРИЛОЖЕНИЙ (RAPID APPLICATION DEVELOPMENT) RAD-ТЕХНОЛОГИЯ

В основе спиральной модели жизненного цикла лежит применение прототипной технологии или RAD-технологии (rapid application development — технологии быстрой разработки приложений). Основная идея этой технологии заключается в том, что ИС разрабатывается путем расширения программных прототипов, повторяя путь от детализации требований к детализации программного кода. При прототипной технологии сокращается число итераций, возникает меньше ошибок и несоответствий, которые необходимо исправлять на последующих итерациях, а само проектирование ИС осуществляется более быстрыми темпами, упрощается создание проектной документации. Для более точного соответствия проектной документации разработанной ИС все большее значение придается ведению общесистемного репозитария и использованию CASE-технологий.

RAD-технология обеспечивает экстремально короткий цикл разработки ИС.

При полностью определенных требованиях и ограниченной проектной области RAD-технология позволяет создать полностью функциональную систему за очень короткое время (60-90 дней).

Выделяют следующие этапы разработки ИС с использованием RAD-технологии:

1. бизнес-моделирование. Моделируется информационный поток между бизнес-функциями. Определяются ответы на вопросы: Какая информация руководит бизнес-процессом? Какая информация генерируется? Кто генерирует ее? Где информация применяется? Кто обрабатывает информацию?
2. моделирование данных. Информационный поток отображается в набор объектов данных, которые требуются для поддержки деятельности организации. Определяются характеристики (свойства, атрибуты) каждого объекта, отношения между объектами;
3. моделирование обработки. Определяются преобразования объектов данных, обеспечивающие реализацию бизнес-функций. Создаются описания обработки для добавления, модификации, удаления или нахождения (исправления) объектов данных;

4. генерация приложения. Предполагается использование методов, ориентированных на языки программирования 4-го поколения. Вместо создания ПО с помощью языков программирования 3-го поколения, RAD-процесс работает с повторно используемыми программными компонентами или создает повторно используемые компоненты. Для обеспечения конструирования используются утилиты автоматизации (CASE-средства);
5. тестирование и объединение. Поскольку применяются повторно используемые компоненты, многие программные элементы уже протестированы, что сокращает время тестирования (хотя все новые элементы должны быть протестированы).

Применение RAD имеет и свои недостатки, и ограничения:

- большие проекты в RAD требуют существенных людских ресурсов (необходимо создать достаточное количество групп);
- RAD применима только для приложений, которые можно разделять на отдельные модули и в которых производительность не является критической величиной;
- RAD неприменима в условиях высоких технических рисков.

Улучшение процесса принятия решений в планировании, разработке, эксплуатации

RAD-технология обеспечивается наличием средств разработки графического пользовательского интерфейса и кодогенераторов. Такие инструментальные средства, как Oracle Designer/2000, JavaJbuilder 3, Linux, Visual C++, Visual Basic 6, SAS, и другие можно использовать в качестве средств для быстрой разработки приложений.

Инкрементная модель жизненного цикла разработки ПО

Инкрементная разработка представляет собой процесс частичной реализации всей системы и медленного наращивания функциональных возможностей. Этот подход позволяет уменьшить затраты, понесенные до момента достижения уровня исходной производительности. С помощью этой модели ускоряется процесс создания функционирующей системы. Этому способствует применяемый принцип компоновки из стандартных блоков, благодаря которому обеспечивается контроль над процессом разработки изменяющихся требований.

Инкрементная модель действует по принципу каскадной модели с перекрытиями, благодаря чему функциональные возможности продукта, пригодные к эксплуатации, формируется раньше. Для этого может понадобиться полный заранее сформированный набор требований, которые выполняются в виде последовательных, небольших по размеру проектов, либо выполнение проекта может начаться с формулирования общих целей, которые затем уточняются и реализуются группами разработчиков.

Подобное усовершенствование каскадной модели одинаково эффективно при использовании как в случае чрезвычайно больших, так и небольших проектов.

Например, на инкременте 1 определяются базовые алгоритмы и выходные данные, на инкременте 2 добавляются некоторые ценные возможности производственного типа, такие как возможность занесения в файл и выборки результатов предыдущих прогонов программы, а на инкременте 3 добавляются различные полезные свойства к пользовательскому интерфейсу, а также к заранее определенным вычислительным свойствам системы.

Инкрементная модель описывает процесс, при выполнении которого первостепенное внимание уделяется системным требованиям, а затем их реализации в группах разработчиков. Как правило, со временем инкременты уменьшаются и реализуют каждый раз меньшее количество требований. Каждая последующая версия системы добавляет к предыдущей определенные функциональные возможности до тех пор, пока не будут реализованы все запланированные возможности. В этом случае можно уменьшить затраты, контролировать влияние изменяющихся требований и ускорить создание функциональной системы благодаря использованию метода компоновки из стандартных блоков.

Фазы инкрементной модели ЖЦ разработки ПО

Предполагается, что на ранних этапах жизненного цикла (планирование, анализ и разработка проекта) выполняется конструирование системы в целом. На этих этапах определяются относящиеся к ним инкременты и функции.
Каждый инкремент затем проходит через остальные фазы жизненного цикла: кодирование, тестирование и поставку.
Сначала выполняется конструирование, тестирование и реализация набора функций, формирующих основу продукта, или требований первостепенной важности, играющих основную роль для успешного выполнения проекта либо снижающих степень риска. Последующие итерации распространяются на ядро системы, постепенно улучшая ее функциональные возможности или рабочую характеристику. Добавление функций осуществляется с помощью выполнения существенных инкрементов с целью в комплексного удовлетворения потребностей пользователя. Каждая дополнительная функция аттестуется в соответствии с целым набором требований.

Преимущества инкрементной модели

Применяя инкрементную модель при разработке проекта, для которого она подходит в достаточной мере, можно убедиться в следующих ее преимуществах:

- не требуется заранее тратить средства, необходимые для разработки всего проекта (поскольку сначала выполняется разработка и реализация основной функции или функции из группы высокого риска);
- в результате выполнения каждого инкремента получается функциональный продукт;
- заказчик располагает возможностью высказаться по поводу каждой разработанной версии системы;
- правило по принципу "разделяй и властвуй" позволяет разбить возникшую проблему на управляемые части, благодаря чему предотвращается формирование громоздких перечней требований, выдвигаемых перед командой разработчиков;
- существует возможность поддерживать постоянный прогресс в ходе выполнения проекта;

- снижаются затраты на первоначальную поставку программного продукта;
- ускоряется начальный график поставки (что позволяет соответствовать возросшим требованиям рынка);
- снижается риск неудачи и изменения требований;
- заказчики могут распознавать самые важные и полезные функциональные возможности продукта на более ранних этапах разработки;
- риск распределяется на несколько меньших по размеру инкрементов (не сосредоточен в одном большом проекте разработки);
- требования стабилизируются (посредством включения в процесс пользователей) на момент создания определенного инкремента, поскольку не являющиеся особо важными изменения отодвигаются на момент создания последующих инкрементов;

- инкременты функциональных возможностей несут больше пользы и проще при тестировании, чем продукты промежуточного уровня при поуровневой разработке по принципу "сверху-вниз"
- улучшается понимание требований для более поздних инкрементов (что обеспечивается благодаря возможности пользователя получить представление о раннее полученных инкрементов на практическом уровне);
- в конце каждой инкрементной поставки существует возможность пересмотреть риски, связанные с затратами и соблюдением установленного графика;
- использование последовательных инкрементов позволяет объединить полученный пользователями опыт в виде усовершенствованного продукта, затратив при этом намного меньше средств, чем требуется для выполнения повторной разработки;

- в процессе разработки можно ограничить количество персонала таким образом, чтобы над поставкой каждого инкремента последовательно работала одна и та же команда и все задействованные в процессе разработки команды не прекращали работу над проектом (график распределения рабочей силы может выравниваться посредством распределения по времени объема работы над проектом);
- возможность начать построение следующей версии проекта на переходном этапе предыдущей версии сглаживает изменения, вызванные сменой персонала;
- в конце каждой инкрементной поставки существует возможность пересмотреть риски, связанные с затратами и соблюдением установленного графика;

- потребности клиента лучше поддаются управлению, поскольку время разработки каждого инкремента очень незначительно;
- поскольку переход из настоящего в будущее не происходит моментально, заказчик может привыкать к новой технологии постепенно;
- ощутимые признаки прогресса при выполнении проекта помогают поддерживать вызванное соблюдением графика "давление" на управляемом уровне.

Недостатки инкрементной модели

При использовании этой модели относительно проекта, для которого она подходит не в достаточной мере, проявляются следующие недостатки:

- в модели не предусмотрены итерации в рамках каждого инкремента;
- определение полной функциональной системы должно осуществляться в начале жизненного цикла, чтобы обеспечить определение инкрементов;
- формальный критический анализ и проверку намного труднее выполнить для инкрементов, чем для системы в целом;
- заказчик должен осознавать, что общие затраты на выполнение проекта не будут снижены;

- поскольку создание некоторых модулей будет завершено значительно раньше других, возникает необходимость в четко определенных интерфейсах;
- использование на этапе анализа общих целей, вместо полностью сформулированных требований, может оказаться неудобным для руководства;
- для модели необходимы хорошее планирование и проектирование: руководство должно заботиться о распределении работы, а технический персонал должен соблюдать субординацию в отношениях между сотрудниками.
- может возникнуть тенденция к оттягиванию решений трудных проблем на будущее с целью продемонстрировать руководству успех, достигнутый на ранних этапах разработки;

Область применения инкрементной модели

Менеджер проекта может быть уверен в целесообразности применения модели, если для этого имеются следующие причины:

- если большинство требований можно сформулировать заранее, но их появление ожидается через определенный период времени;
- если рыночное окно слишком "узкое" и существует потребность быстро поставить на рынок продукт, имеющий функциональные базовые свойства;
- для проектов, на выполнение которых предусмотрен большой период времени разработки, как правило, один год;
- при равномерном распределении свойств различной степени важности;

- когда при рассмотрении риска, финансирования, графика выполнения проекта, размера программы, ее сложности или необходимости в реализации на ранних фазах оказывается, что самым оптимальным вариантом является применение принципа пофазовой разработки;
- при разработке программ, связанных с низкой или средней степенью риска;
- при выполнении проекта с применением новой технологии, что позволяет пользователю адаптироваться к системе путем выполнения более мелких инкрементных шагов, без резкого перехода к применению основного нового продукта;
- когда однопроходная разработка системы связана с большой степенью риска;
- когда результативные данные получаются через регулярные интервалы времени.

Выбор приемлемой модели жизненного цикла ПО ИС

Выбор приемлемой модели жизненного цикла разработки ПО для проекта может осуществляться в ходе использования следующего процесса.
Проанализируйте следующие отличительные категории проекта, помещенные в таблицах 1-4:
Требования: таблица 1.
Команда разработчиков: таблица 2.
Коллектив пользователей: таблица 3.
Тип проекта и риски: таблица 4.
Ответьте на вопросы, приведенные для каждой категории, обведя кружочком слова "да" или "нет".

Расположите по степени важности категории или вопросы, относящиеся к каждой категории, относительно проекта, для которого выбирается приемлемая модель.
Воспользуйтесь упорядоченными категориями для разрешения противоречий, возникающих при сравнении моделей, если общие полученные показатели сходны или одинаковы.

В табл. 1-4 приведен набор матриц, предназначенных для выбора модели жизненного цикла.
Требования. Категория требований (таблица 1) состоит из вопросов относительно требований, которые предъявляет пользователь к проекту. В терминологии их иногда называют свойствами системы, которая будет поддерживаться данным проектом.

Таблица 1. Выбор модели жизненного цикла на основе характеристик требований

Требования	Каскад-Ная	V-образ-ная	Прототи-пирование	Спираль-ная	RAD	Инкре-ментная
Являются ли требования легко определимыми и/или хорошо известными?	Да	Да	Нет	Нет	Да	Нет
Могут ли требования заранее определяться в цикле?	Да	Да	Нет	Нет	Да	Да
Часто ли будут изменяться требования в цикле?	Нет	Нет	Да	Да	Нет	Нет
Нужно ли демонстрировать требования с целью определения?	Нет	Нет	Да	Да	Да	Нет
Требуются ли для демонстрации возможностей проверка концепции?	Нет	Нет	Да	Да	Да	Нет
Будут ли требования отражать сложность системы?	Нет	Нет	Да	Да	Нет	Да
Обладает ли требование функциональными свойствами на раннем этапе?	Нет	Нет	Да	Да	Да	Да

Вывод: если требования не могут быть заранее определены, а в ходе работ будут часто изменяться, то наиболее подходящими являются модель прототипирования и спиральная модель.

Команда разработчиков. По возможности, в состав команды разработчиков лучше всего отобрать персонал еще до того, как будет выбрана модель жизненного цикла. Характеристики такой команды (таблица 4.2) играют важную роль в процессе выбора, поскольку она несет ответственность за удачное выполнение цикла и может оказать помощь в процессе выбора.

Таблица 2. Выбор модели жизненного цикла на основе характеристик участников команды разработчиков

Команда разработчиков проекта	Каскад-ная	V-образ-ная	Прототи пирован	Спираль ная	RAD	Инкре-ментная
Являются ли проблемы предметной области проекта новыми для большинства разработчиков?	Нет	Нет	Да	Да	Нет	Нет
Является ли технология предметной области проекта новой для большинства разработчиков?	Да	Да	Нет	Да	Нет	Да
Являются ли инструменты, используемые проектом, новыми для большинства разработчиков?	Да	Да	Нет	Да	Нет	Нет
Изменяются ли роли участников проекта во время жизненного цикла?	Нет	Нет	Да	Да	Нет	Да
Могут ли разработчики проекта пройти обучение?	Нет	Да	Нет	Нет	Да	Да
Является ли структура более значимой для разработчиков, чем гибкость?	Да	Да	Нет	Нет	Нет	Да
Будет ли менеджер проекта строго отслеживать прогресс команды?	Да	Да	Нет	Да	Нет	Да
Важна ли легкость распределение ресурсов?	Да	Да	Нет	Нет	Да	Да
Приемлет ли команда равноправные обзоры и инспекции, менеджмент/обзоры заказчика, также стадии?	Да	Да	Да	Да	Нет	Да

Коллектив пользователей. На начальных фазах проекта можно получить четкое представление о коллективе пользователей (табл. 3) и его будущей взаимосвязи с командой разработчиков на протяжении всего проекта. Такое представление поможет вам при выборе подходящей модели, поскольку некоторые модели требуют усиленного участия пользователей в процессе разработки и изучения проекта.

Таблица 3. Выбор модели жизненного цикла на основе характеристик коллектива пользователей

Коллектив пользователей	Каскад-ная	V-образ-ная	Прототи-пирование	Спираль-ная	RAD	Инкре-ментная
Будет ли присутствие пользователей ограничено в жизненном цикле?	Да	Да	Нет	Да	Нет	Да
Будут ли пользователи знакомы с определением системы?	Нет	Нет	Да	Да	Нет	Да
Буду ли пользователи ознакомлены с проблемами предметной области?	Нет	Нет	Да	Нет	Да	Да
Будут ли пользователи вовлечены во все фазы жизненного цикла?	Нет	Нет	Да	Нет	Да	Нет
Будет ли заказчик отслеживать ход выполнения проекта?	Нет	Нет	Да	Да	Нет	Нет

Тип проекта и риски. И, наконец, уточним, что собой представляют тип проекта и риски (таблица 4), которые были рассмотрены как элементы, определение которых осуществляется на фазе планирования. В некоторых моделях предусмотрен менеджмент рисков высокой степени, в то время как в других он не предусмотрен вообще. Выбор модели, которая делает возможным менеджмент рисков, не означает, что вам не нужно составлять план действий, направленный на минимизацию выявленных рисков. Такая модель просто обеспечивает схему, в рамках которой можно обсудить и выполнить данный план действий.

Таблица 4. Выбор модели жизненного цикла на основе характеристик типа проектов и рисков

Тип проекта и риски	Каскад-ная	V-образ-ная	Прототи-пирование	Спираль-ная	RAD	Инкре-ментная
Будет ли проект идентифицировать новое направление продукта для организации?	Нет	Нет	Да	Да	Нет	Да
Будет ли проект иметь тип системной интеграции?	Нет	Да	Да	Да	Да	Да
Будет ли проект являться расширением существующей системы?	Нет	Да	Нет	Нет	Да	Да
Будет ли финансирование проекта стабильным на всем протяжении жизненного цикла?	Да	Да	Да	Нет	Да	Нет
Ожидается ли длительная эксплуатация продукта в организации?	Да	Да	Нет	Да	Нет	Да
Должна ли быть высокая степень надежности?	Нет	Да	Нет	Да	Нет	Да
Будет ли система изменяться, возможно, с применением непредвиденных методов, на этапе сопровождения?	Нет	Нет	Да	Да	Нет	Да
Является ли график ограниченным?	Нет	Нет	Да	Да	Да	^ J
Являются ли "прозрачными" интерфейсные модули?	Да	Да	Нет	Нет	Нет	Да
Доступны ли повторное используемые компоненты?	Нет	Нет	Да	Да	Да	Нет
Являются ли достаточными ресурсы (время, деньги, инструменты, персонал)?	Нет	Нет	Да	Да	Нет	Нет

ВЫВОДЫ

Существует множество различных моделей или представлений жизненного цикла разработки ПО. Все они представляют собой логически построенную последовательность действий, начиная с определения потребности и заканчивая производством ПО. Каждая модель представляет собой процесс, который структурно состоит из этапов, направленных на обеспечение целостности соответствующих субкомпонентных действий. Каждая фаза снижает степень риска при выполнении проекта, что достигается благодаря применению критериев входа и выхода для определения дальнейшего хода действий. По завершении каждой фазы получают внутренние или результативные внешние действия.

Жизненные циклы разработки ПО иногда называют методиками менеджмента жизненных циклов. Эти методики охватывают все стандарты и процедуры, оказывающие влияние на планирование, сбор требований и анализ, разработку проекта, конструирование и внедрение программной системы. С целью обеспечения эффективности произвольного жизненного цикла его потребуется аккуратно выбрать и зачастую настроить (подогнать и разработать) в соответствии с задачами и целями определенного проекта. Вместо того чтобы начать разработку "с нуля", в некоторых популярных, обобщенных моделях обеспечиваются готовые начальные схемы. Каждая модель имеет присущие ей преимущества и недостатки, определяющие ее применение для определенных типов проектов.

Модель, выбранная для какого-либо проекта, должна обеспечивать потребности организации, соответствовать типу выполняемых работ, а также навыкам и инструментальным средствам, которые имеются у специалистов-практиков.

Убедившись в эффективности использования моделей жизненного цикла в рамках процесса, вы можете помочь вашей организации достичь гибкости при выполнении проекта. В каждом проекте, выполняемом организацией, можно применить отдельную модель жизненного цикла, которая подвергается настройке.

Однако интеграция моделей жизненного цикла с "каркасом" процесса — это уже другая стадия в ходе достижения более высокого уровня завершенности процесса разработки ПО.

Организация должна осознать то, что разрабатываемые программы должны обладать постоянными характеристиками. В то же время реализация этого процесса должна быть гибкой, что обеспечивается с помощью настраиваемых моделей жизненного цикла разработки ПО.

Из приведенных рассуждений следует, процедура выбора модели жизненного цикла должна осуществляться на основе рассмотрения не отдельных критериев, а их комплекса. При этом существенную роль будут играть особенности реальной ситуации.