Лекция 09

Введение в разработку программного обеспечения (ИС и ЦД)

Технологии разработки ПО. Управление проектом.
Стандарты программной инженерии

План лекции:
- управление проектом;
- управление командой.
- профессиональные и этические требования;
- стандарты программной инженерии.

1. Методология гибкой разработки программного обеспечения Agile

Agile – семейство гибких итеративно-инкрементальных методов управления проектами и продуктами.

Идеи Agile

- ✓ Люди и их ***взаимодействие*** важнее, чем процессы и инструменты
- ✓ ***Рабочее*** ПО важнее, чем документация
- ✓ ***Клиенты и сотрудничество*** с ними важнее, чем контракт и обсуждение условий
- ✓ *Готовность к* ***внесению изменений*** важнее, чем первоначальный план

Принципы Agile

- ✓ ***Удовлетворять клиентов, заблаговременно и постоянно поставляя ПО (клиенты довольны, когда рабочее ПО поступает к ним регулярно и через одинаковые промежутки времени)***
- ✓ ***Изменять требования к конечному продукту в течение всего цикла его разработки***
- ✓ ***Поставлять рабочее ПО как можно чаще (раз в неделю, в две недели, в месяц и т.д.)***
- ✓ ***Поддерживать сотрудничество между разработчиками и заказчиком в течение всего цикла разработки***
- ✓ ***Поддерживать и мотивировать всех, кто вовлечен в проект (если команда мотивирована, она намного лучше справляется со своими задачами, нежели команда, члены которой условиями труда недовольны)***
- ✓ ***Обеспечивать непосредственное взаимодействие между разработчиками (возможность прямого контакта способствует более успешной коммуникации)***
- ✓ ***Измерять прогресс только посредством рабочего ПО (клиенты должны получать только функциональное и рабочее программное обеспечение)***
- ✓ ***Поддерживать непрерывный темп работы (команда должна выработать оптимальную и поддерживаемую скорость работы)***
- ✓ ***Уделять внимание дизайну и техническим деталям (благодаря эффективным навыкам и хорошему дизайну команда проекта получает возможность постоянного совершенствования продукта и работы над его улучшением)***
- ✓ ***Стараться сделать рабочий процесс максимально простым, а ПО – простым и понятным***
- ✓ ***Позволять членам команды самостоятельно принимать решения (если разработчики могут сами принимать решения, самоорганизовываться и общаться с другими членами коллектива, обмениваясь с ними идеями, вероятность создания качественного продукта существенно возрастает)***
- ✓ ***Постоянно адаптироваться к меняющейся среде (благодаря этому конечный продукт будет более конкурентоспособен)***

в 2001 году в штате Юта (США) на курорте Snowbird собрались семнадцать разработчиков программного обеспечения. В результате обсуждения методов разработки был опубликован «Манифест о гибкой разработке программного

обеспечения Agile» (в переводе с английского понятие «agile» означает «подвижный», «проворный» или «быстрый», но в большинстве случаев его переводят именно как «гибкий»). Он и задал темп всей дальнейшей работе над созданием ПО.

Манифест Agile

Манифест, созданный программистами, включает в себя 4 базовых идеи и 12 принципов эффективного управления проектами. Любая из систем управления проектами на основе Эджайл (о системах мы поговорим позже) опирается именно на эти идеи и принципы, хотя и использует их в разных вариацияУ человечества за всю историю накопился внушительный список успешно реализованных сложных проектов. От строительства Пирамид в Гизе до отправки человека на Луну, самые

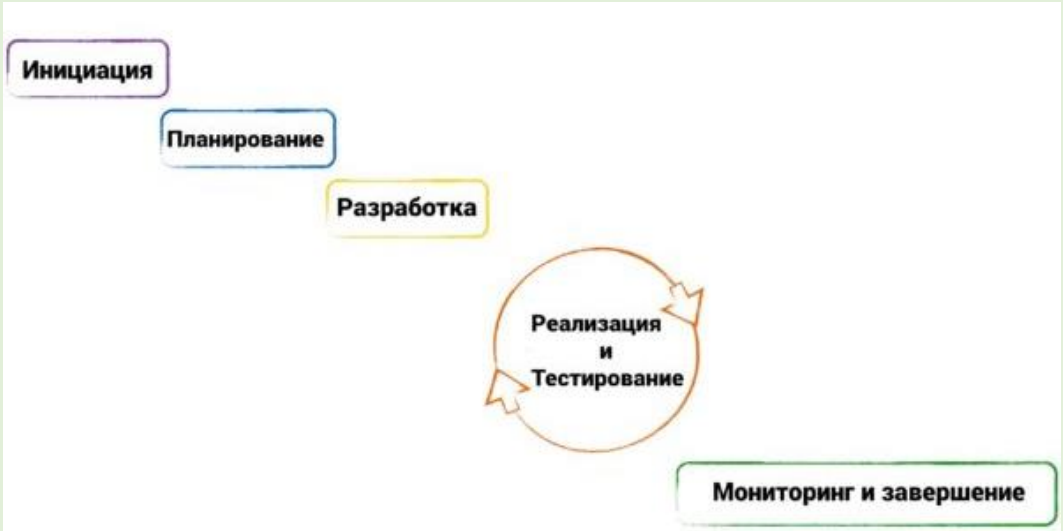
> ***«Из всех трудностей, с которыми столкнулись НАСА, отправляя человека на Луну, управление было наверно самой сложной задачей»***
>
> ***— Роджер Лаунис, историк НАСА***

смелые человеческие начинания требовали слаженной работы тысяч людей. А это подразумевает сложную систему управления проектами.

> ***Управление проектами*** – это управление и организация процесса разработки ПО для достижения поставленной цели в установленное время и в рамках бюджета.

«Классическое» или «традиционное» проектное управление: Наиболее широко распространённый метод управления проектами, основанный на так называемом «водопадном» (Waterfall) или каскадном цикле, при котором задача передаётся последовательно по этапам

> ***Классическое управление проектами*** – это метод управления проектами, основанный на водопадной модели ЖЦ ПО (Waterfall).



Подходит для типовых проектов: – проекты с большим количеством задач; – взаимосвязей; – дедлайнов; – видов ресурсов.	***Простейший способ планирования реализации проекта –*** разбить его на фазы или отдельные задачи. ***Простейший инструмент проектного управления –*** список (чек-лист) действий, необходимых для достижения цели.

Управление проектами – это управление и организация всего, что нужно для достижения цели – вовремя и в рамках бюджета, конечно же. Будь до разработка нового программного обеспечения, проведение маркетинговой компании или высадка человека на Марс – проектное управление позволяет добиться успеха.

Самый очевидный путь реализации проекта – разбить его на фазы или отдельные задачи.

Простейший инструмент проектного управления представляет собой чек-лист действий, которые необходимо совершить для достижения цели. Просто и эффективно.

кулинарный рецепт – покупаете ингредиенты, правильно их смешиваете, готовите и подаёте.

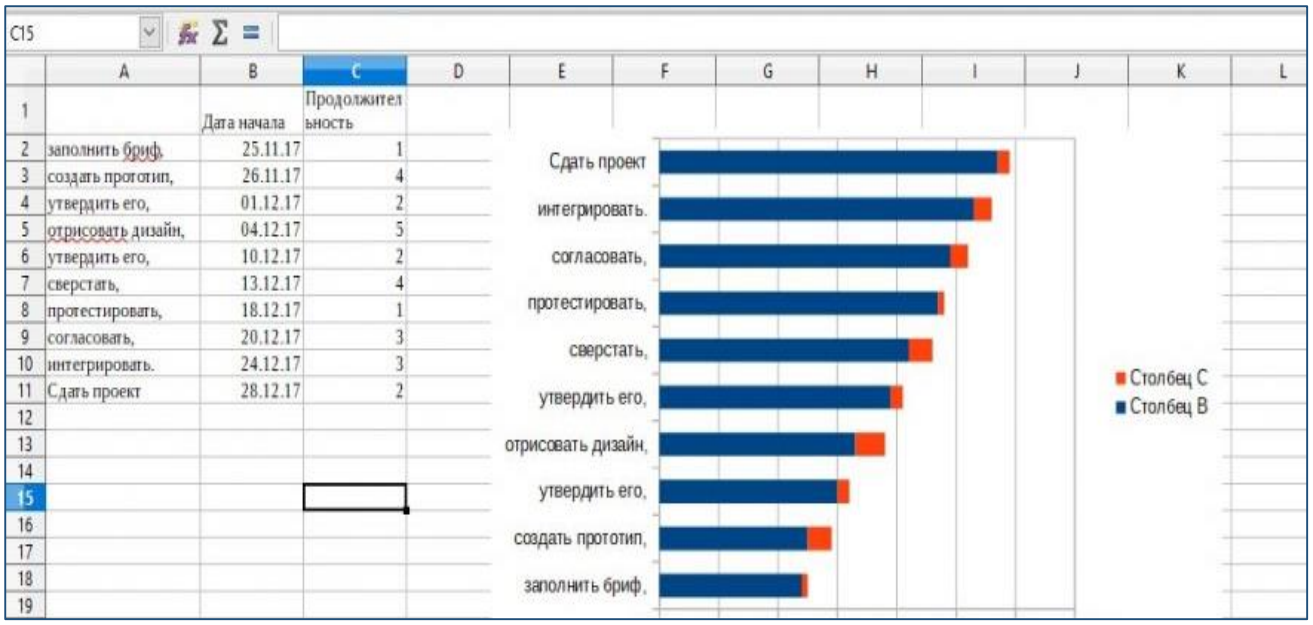
Однако, если Вы – шеф-повар, и готовите не одно блюдо, а несколько, например, салат (приготовление которого состоит из 3 этапов) и десерт (который нужно только подать), то Вам потребуется инструмент, позволяющий отслеживать временные затраты на каждый из элементов и время, когда они должны быть готовы.

И тут на помощь приходит один из первых современных инструментов проектного управления: Диаграмма Гантта, представленная на Рисунке 2.

Инструменты проектного управления

Диаграмма Гантта:

- ✓ подходит для проектов с жесткими ресурсными ограничениями;
- ✓ предоставляет возможность построить схему в форме событийной цепочки процессов (event-driven process chain) для планирования ресурсов.



Изобретённая независимо Королем Адамеки (Korol Adamecki) и Генри Л. Ганттом (Genry L. Gantt) в начале XX в., диаграмма Гантта показывает расписание проекта основываясь на датах окончания и завершения задач. В неё вносятся задачи, их длительности и взаимосвязи, а затем высчитывается критический путь – самая длинная цепочка взаимосвязанных задач, определяющих длительность проекта. Взаимосвязи между началом и окончанием разных задач очень важны – вы же не можете подать гостям суп, пока вы его не сварили, не так ли?

Подходит для типовых проектов

в нём гораздо больше задач, взаимосвязей, дедлайнов и видов ресурсов. Проектам с жёсткими дедлайнами диаграмма Гантта помогает решить, когда лучше начинать те или иные задачи, чтобы сократить время реализации. А для проектов с сильными ресурсными ограничениями, диаграмма Гантта предоставляет возможность построить схему в форме событийной цепочки процессов (event-driven process chain) для планирования ресурсов.

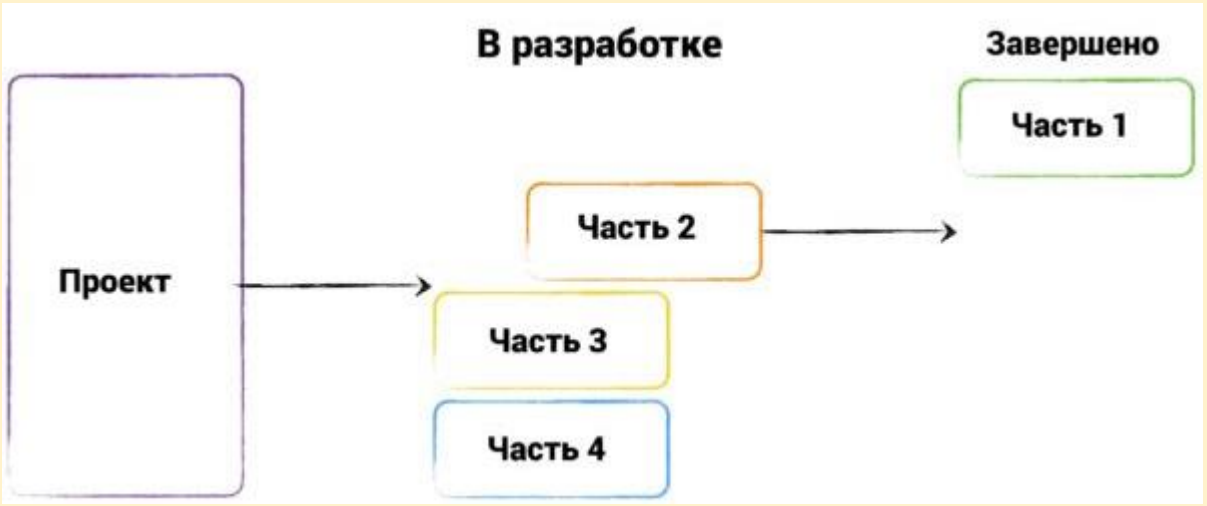
Разным проектам нужен различный уровень контроля. Например, если вы публикуете серию статей в блоге, то, жёсткие дедлайны не так важны. Гораздо важнее чёткий процесс, в рамках которого есть возможность составить структуру каждой статьи, сделать набросок каждой из них, получить обратную связь, внести правки, закончить статью, вычитать и опубликовать. Вместо управления временем и ресурсами, вы управляете процессом.

Для таких проектов лучше подходят гибкие методы управления проектами Agile и связанные с ним подходы, такие как Lean, Kanban и другие. Есть и методы, позволяющие управлять как рабочим потоком, так и временем, и ресурсами

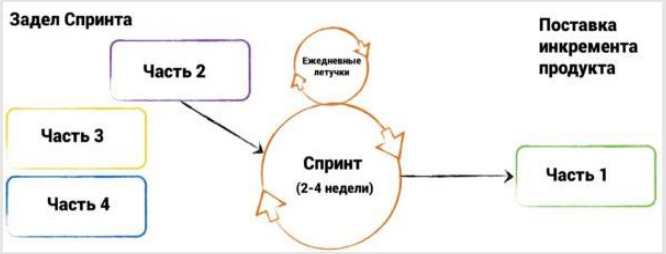
Гибкий итеративно-инкрементальный подход к управлению проектами

Гибкий итеративно-инкрементальный подход к управлению проектами

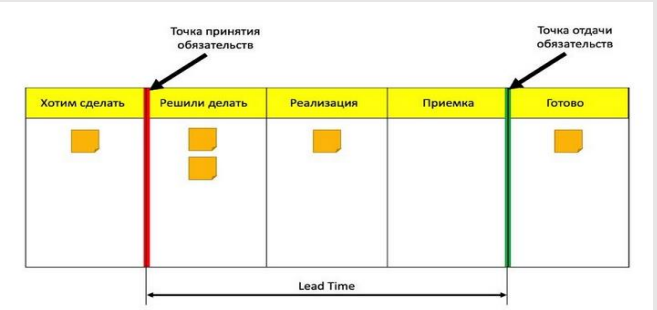
- ✓ ориентирован на динамическое формирование требований;
- ✓ обеспечение их реализации;
- ✓ постоянного взаимодействия внутри самоорганизующихся рабочих групп; группы состоят из специалистов различного профиля.



метод Scrum:



метод Kanban



Agile: Гибкий итеративно-инкрементальный подход к управлению проектами и продуктами, ориентированный на динамическое формирование требований и обеспечение их реализации в результате постоянного взаимодействия внутри самоорганизующихся рабочих групп, состоящих из специалистов различного профиля. Существует множество методов, базирующихся на идеях Agile, самые популярные из которых – Scrum и Kanban.

Критический путь: Непрерывная последовательность работ и событий от начального до конечного события, требующая наибольшего времени для её выполнения.

Событийная цепочка процессов (EPC-диаграмма): диаграмма, отображающая последовательность реализации работ проектов основываясь на доступности и загруженности ресурсов

Резерв времени: Время, на которое может быть отложено начало работы без влияния на общую продолжительность проекта. Таким образом, у работ на критическом пути резерв будет равняться нулю.

Веха (контрольная точка, milestone): Ключевое событие, обозначающее, например, конец этапа. На диаграмме Гантта обозначается задачей с нулевой длительностью.

Менеджер проекта (руководитель проекта, project manager, PM): Руководитель команды проекта, ответственный за управление проектом (планирование, реализацию и закрытие проекта).

Ресурсы: Элементы, необходимые для реализации проекта. Ресурсами являются время, оборудование, материалы, сотрудники и прочее.

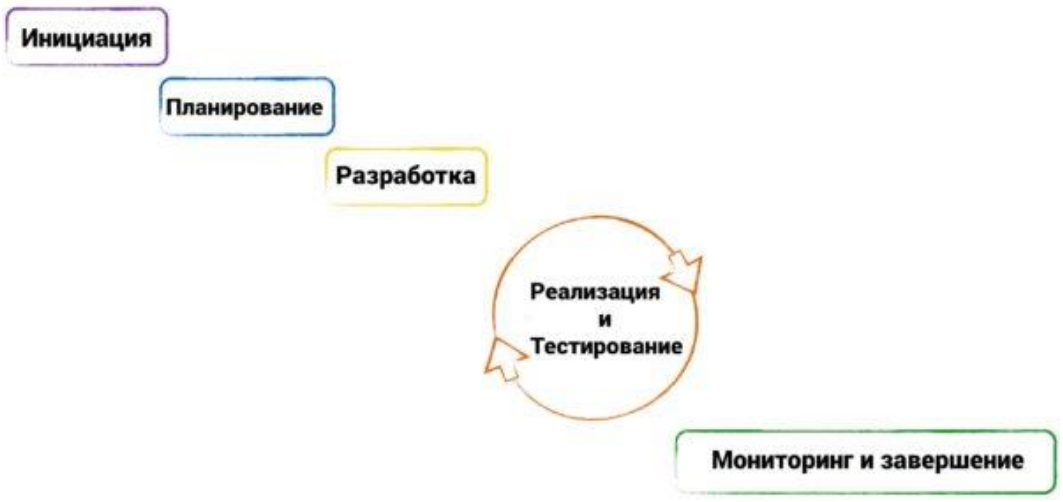
Содержание проекта (Scope): Описание работ, которые необходимо выполнить, чтобы получить продукт.

Спринт (Sprint): Итерация (рабочий цикл) в Scrum, длящаяся от недели до месяца, в ходе которой создаётся рабочая версия продукта или его элемент, представляющий ценность для заказчика.

«Классическое» или «традиционное» проектное управление: Наиболее широко распространённый метод управления проектами, основанный на так называемом «водопадном» (Waterfall) или каскадном цикле, при котором задача передаётся последовательно по этапам, напоминающим поток.

Классическое проектное управление

Рисунок 3: Схема Классического проектного подхода



Данный подход ориентирован на проекты, в которых есть строгие ограничения по последовательности выполнения задач.

Обычно выделяют 5 этапов классического проектного управления, но можно добавлять и дополнительные этапы, если того требует проект.

5 этапов традиционного менеджмента:

Этап 1. Инициация. Руководитель проекта и команда определяют требования к проекту. На данном этапе часто проводятся совещания и «мозговые штурмы», на которых определяется что же должен представлять из себя продукт проекта.

Этап 2. Планирование. На данном этапе команда решает, как она будет достигать цели, поставленной на предыдущем этапе. На данном этапе команда уточняет и детализует цели и результаты проекта, а также состав работ по нему. На основании данной информации команда формирует календарный план и бюджет, оценивает риски и выявляет заинтересованные стороны.

Этап 3. Разработка. Данная стадия реализуется не для всех проектов — как правило она является частью фазы планирования. В фазе разработки, характерной для технологических проектов, определяется конфигурация будущего проекта и/или продукта и технические способы его достижения. Например в ИТ-проектах на данном этапе выбирается язык программирования. (В отечественной практике данная фаза обычно не выделяется, а термин «разработка» не используется — прим. пер.)

Этап 4. Реализация и тестирование. На этой фазе происходит собственно основная работа по проекту – написание кода, возведение здания и тому подобное. Следуя разработанным планам начинает создаваться содержание проекта, определённое ранее, проводится контроль по выбранным метрикам. Во второй части данной фазы происходит тестирование продукта, он проверяется на соответствие требованиям Заказчика и заинтересованных сторон. В части тестирования выявляются и исправляются недостатки продукта.

Этап 5. Мониторинг и завершение проекта. В зависимости от проекта данная фаза может состоять из простой передачи Заказчику результатов проекта или же из длительного процесса взаимодействия с клиентами по улучшению проекта и повышению их удовлетворённости, и поддержке результатов проекта. Последнее относится к проектам в области клиентского сервиса и программного обеспечения.

Это выше – база, на которой строятся различные методы управления проектами.

проект разбит на этапы, которые исполняются в строго определённой последовательности.

Благодаря тому, что классический проектный менеджмент строго привязан ко времени исполнения задач, как правило, заранее определённому на этапе планирования, для реализации проектов в рамках данного подхода отлично подходят инструменты календарно-сетевого планирования. Самым распространённым инструментом календарно-сетевого планирования является уже упомянутая ранее диаграмма Гантта. Существует множество инструментов для её построения – от простых таблиц вроде Excel и Smartsheet до профессиональных программных пакетов вроде Microsoft Project и Primavera.

+

плюсом данного подхода является то, что он требует от Заказчика и руководства компании определить, что же они хотят получить, уже на первом этапе проекта. Раннее включение привносит определённую стабильность в работу проекта, а планирование позволяет упорядочить реализацию проекта. Кроме того, этот подход подразумевает мониторинг показателей и тестирование, что совершенно необходимо для реальных проектов различного масштаба.

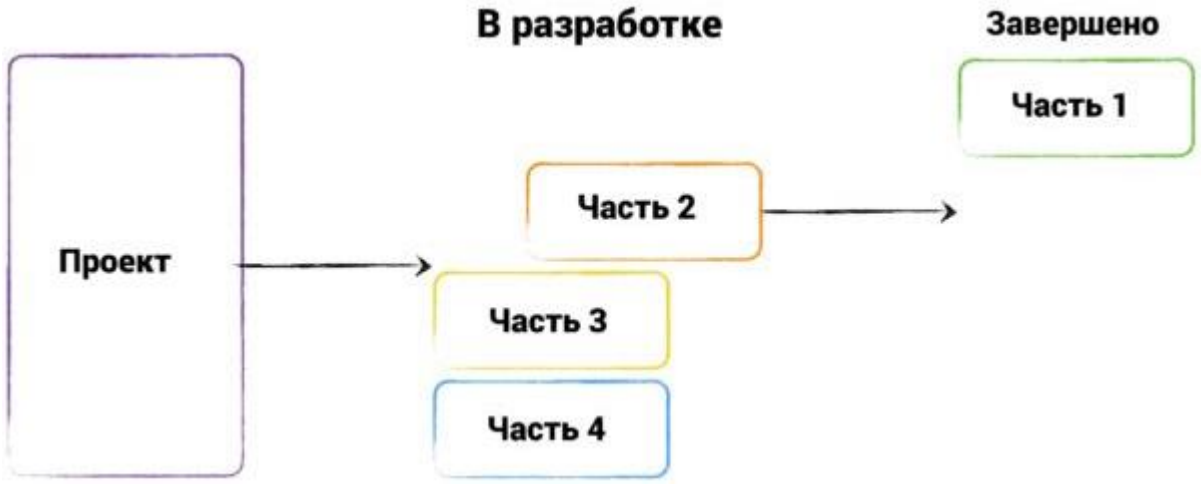
классический подход позволяет избежать стрессов ввиду наличия запасного времени на каждом этапе, заложенного на случай каких-либо осложнений и реализации рисков. Кроме того, с правильно проведённым этапом планирования, руководитель проектов всегда знает, какими ресурсами он обладает. Даже если эта оценка не всегда точная.

-

слабая сторона классического проектного менеджмента – недостаток гибкости.

Agile – семейство гибких итеративно-инкрементальных методов к управлению проектами и продуктами. Согласно данному подходу, проект разбивается не на последовательные фазы, а на маленькие подпроекты, которые затем «собираются» в готовый продукт. Схема работы

Таким образом, инициация и верхнеуровневое планирование проводятся для всего проекта, а последующие этапы: разработка, тестирование и прочие проводятся для каждого мини-проекта отдельно. Это позволяет передавать результаты этих мини-проектов, так называемые, инкременты, быстрее, а приступая к новому подпроекту (итарации) в него можно внести изменения без больших затрат и влияния на остальные части проекта.



2001 с публикации Манифеста Agile (Agile Manifesto), закрепившем основные ценности и принципы гибкой разработки программного обеспечения, в основе которых – командная работа и адаптация, даже «любовь» к изменениям.

на основе этих принципов и лучших практик были разработаны отдельные гибкие методы или, как их иногда называют, фреймворки (frameworks): Scrum, Kanban, Crystal, и многие другие.

+

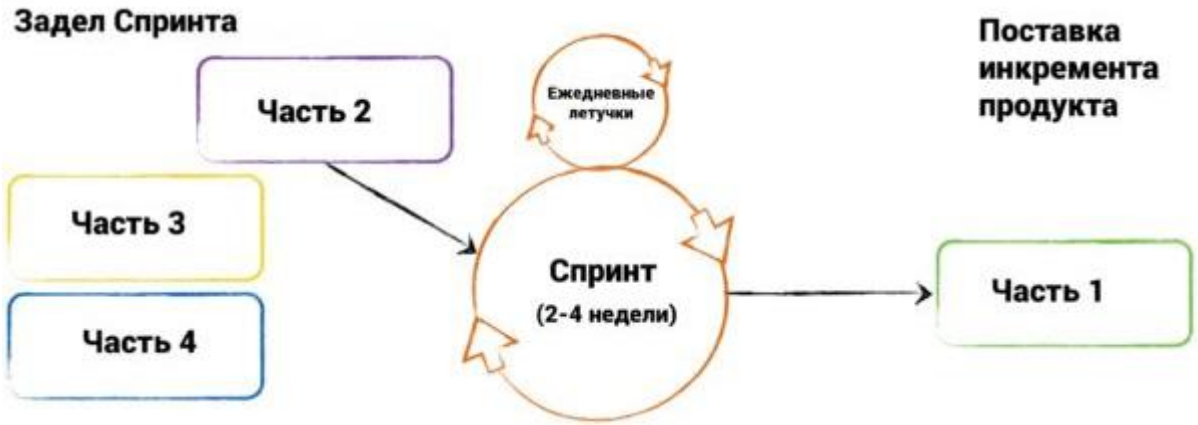
гибкость и адаптивность

подходит для разработки новых, инновационных продуктов. В проектах по разработке таких продуктов высока доля неопределённости, а информация о продукте раскрывается по ходу проекта.

-

это набор принципов и ценностей. Слабая сторона состоит в том, что каждой команде придётся самостоятельно составлять свою систему управления, руководствуясь принципами Agile. Это непростой и длительный процесс

Следуя заветам Agile, Scrum разбивает проект на части, которые сразу могут быть использованы Заказчиком для получения ценности, называемые заделами продуктов (product backlog). И несмотря на то, что «задел продукта» — достаточно верный перевод и используется в профессиональной литературе, в российской практике чаще всего используется просто «беклог». Затем эти части приоретизируются Владельцем продукта – представителем Заказчика в команде. Самые важные «кусочки» первыми отбираются для выполнения в Спринте – так называются итерации в Scrum, длящиеся от 2 до 4 недель. В конце Спринта Заказчику представляется рабочий инкремент продукта – те самые важные «кусочки», которые уже можно использовать. Например, сайт с частью функционала или программа, которая уже работает, пусть и частично. После этого команда проекта приступает к следующему Спринту. Длительность у Спринта фиксированная, но команда выбирает её самостоятельно в начале проекта, исходя из проекта и собственной производительности.



Чтобы удостовериться в том, что проект отвечает требованиям Заказчика, которые имеют свойство изменяться со временем, перед началом каждого Спринта происходит переоценка ещё не выполненного содержания проекта и внесение в него

изменений. В этом процессе участвуют все – команда проекта, Scrum Мастер (Scrum Master, лидер команды проекта) и Владелец продукта. И ответственность за этот процесс лежит на всех.

Основная структура процессов Scrum вращается вокруг 5 основных встреч: упорядочивания беклога, планирования Спринта, ежедневных летучек, подведения итогов Спринта и ретроспективы Спринта.

- Встреча по упорядочиванию беклога (Backlog Refinement Meeting, «Backlog Grooming»): Эта встреча аналогична фазе планирования в классическом проектном управлении, и проводится в первый день каждого Спринта. На ней рассматривается – что уже было сделано по проекту в целом, что ещё осталось сделать и принимается решение о том, что же делать дальше. Владелец продукта определяет, какие задачи на данном этапе являются наиболее приоритетными. Данный процесс определяет эффективность Спринта, ведь именно от него него зависит, какую ценность получит Заказчик по итогам спринта.

- Планирование Спринта: После того, как Владелец продукта определил приоритеты, команда совместно решает, что же конкретно они будут делать во время грядущей итерации, как достигнуть поставленной на предыдущей встрече цели. Команды могут применять различные инструменты планирования и оценки на данном этапе, лишь бы они не противоречили принципам и логике Scrum. Планирование Спринта проводится в самом начале итерации, после Встречи по упорядочиванию продукта.

- Ежедневные летучки: Каждый день спринта, в идеале, в одно и то же время, члены команды тратят 15 минут на то, чтобы поделиться информацией о статусе задач и состоянии проекта. На ней не происходит обсуждений проблем или принятия решений – если после встречи возникают вопросы и конфликты, Scrum Мастер и вовлечённые участники обсуждают их отдельно. Летучка же нужна для обмена информации и поддержания всех членов команды в курсе состояния проекта.

- Подведение итогов Спринта: Цель этапа – обследование и адаптация создаваемого продукта. Команда представляет результаты деятельности всем заинтересованным лицам. Основная задача – убедиться, что продукт этапа соответствует ожиданиям участников и согласуется с целями проекта.

- Ретроспектива Спринта: Проводится сразу после Подведения итогов спринта и до планирования следующего спринта. На нём команда выясняет, насколько чётко и слаженно проходил процесс реализации этапа. Обследованию подвергаются возникшие проблемы в работе, методологии и взаимодействии. Именно этот этап позволяет команде провести рефлексию и следующий Спринт провести эффективнее.

Многим Scrum может показаться сложным для внедрения – новый процесс, новые роли, много делегирования и совершенно новая организационная структура. Но это

гибкий и при этом структурированный подход к реализации проектов, который, в отличие от размытых и общих принципов Agile, не позволит работе пойти не в то русло.

+

Scrum был разработан для проектов, в которых необходимы «быстрые победы» в сочетании с толерантностью к изменениям. Кроме того, этот фреймворк подходит для ситуаций, когда не все члены команды имеют достаточный опыт в той сфере, в которой реализуется проект – постоянные коммуникации между членами командами позволяют недостаток опыта или квалификации одних сотрудников за счёт информации и помощи от коллег.

-

Scrum очень требователен к команде проекта. Она должна быть небольшой (5-9 человек) и кроссфункциональной – то есть члены команды должны обладать более чем одной компетенцией, необходимой для реализации проекта. Например разработчик ПО должен обладать познаниями в тестировании и бизнес-аналитике. Делается это для того, чтобы часть команды не «простаивала» на разных этапах проекта, а также для того, чтобы сотрудники могли помогать и подменять друг друга.

члены команды должны быть «командными игроками», активно брать на себя ответственность и уметь самоорганизовываться

Kanban

Созданный инженером компании Toyota Тайичи Оно (Taiichi Ono) в 1953 году, Kanban очень похож на схему промышленного производства

инкремент продукта передаётся вперёд с этапа на этап, а в конце получается готовый к поставке элемент.

Kanban намного менее строгий, нежели Scrum – он не ограничивает время спринтов, нет ролей, за исключением владельца продукта. Kanban даже позволяет члену команды вести несколько задач одновременно, чего не позволяет Scrum. Также никак не регламентированы встречи по статусу проекта – можно делать это как Вам удобно, а можно не делать вообще.

во многом Kanban является визуализацией идеи Agile. Но у Kanban есть 4 столпа, на которых держится вся система:

1. Карточки: Для каждой задачи создаётся индивидуальная карточка, в которую заносится вся необходима информация о задаче. Таким образом, вся нужная информация о задаче всегда под рукой.

2. Ограничение на количество задач на этапе: Количество карточек на одном этапе строго регламентировано. Благодаря этому сразу становится видно, когда в потоке операций возникает «затор», который оперативно устраняется.

3. Непрерывный поток: Задачи из беклога попадают в поток в порядке приоритета. Таким образом, работа никогда не прекращается.

4. Постоянное улучшение («кайзен» (kaizen)): Концепция постоянного улучшения появилась в Японии в конце XX века. Её суть в постоянном анализе производственного процесса и поиске путей повышения производительности.

+

хорошо подходит для достаточно сплочённых команды с хорошей коммуникацией. Но в отличие от Scrum, в Kanban нет установленных чётких дедлайнов, что хорошо подходит для замотивированных и опытных команд.

Точный расчёт нагрузки на команду, правильная расстановка ограничений и концентрация на постоянном улучшении — всё это позволяет Kanban серьёзно экономить ресурсы и укладывать в дедлайны и бюджет. И всё это в сочетании с гибкостью.

-

Kanban лучше всего подходит для команд, навыки членов которых пересекаются друг с другом.

Kanban лучше подходит в тех случаях, когда нет жёстких дедлайнов. Для жёстких дедлайнов лучше подходит классический подход или Scrum.

Создание и управление командой проекта Agile

Назначение проекта

Первое, что должен сделать руководитель проекта, это назначение проекта поделиться с другими членами команды. Создание задания по проекту требует от рулевая группа подумать об этом, обсудить это и прийти к соглашению. Задание должно быть сформулировано так, чтобы члены команды могли распознать свои навыки в нем. Если члены команды принимают задание не только формально, но и от всего сердца, это очень мотивирует.

Ответственность команды

Что произойдет, если члены команды не знают, какова их роль и что они должны делать в проекте?

В командной среде очень важно, чтобы члены команды знали, чего от каждого из них ждут.

Для достижения этого вы можете использовать один начальная встреча организация. На стартовой встрече вы должны:

- Уточните, кто отвечает, и что вы ожидаете от команды.
- Убедитесь, что всем понятно, каковы их роль и обязанности. Эти роли уже определены при формировании команды проекта.

Командные встречи

Командные соглашения являются предварительными условиями и руководящими принципами того, как происходит сотрудничество. Команды проекта не нуждаются в большом количестве соглашений, чтобы хорошо работать вместе, но каждый член команды должен согласиться с этими соглашениями и разделить ответственность за их соблюдение.

Командная мотивация

Работа в команде без мотивации - это как тело без души. Если Руководитель проекта Это ваша задача - постоянно мотивировать вашу команду каждый день.

Наличие мотивированной проектной команды улучшит качество и производительность проекта. Создайте хорошую рабочую среду, которая вдохновляет членов команды.

Управление гибкой командой проекта

Управление командой проекта - очень сложная задача для менеджера проекта. Вы имеете дело с разными личностями и разными стилями работы. Основная цель хорошего менеджера - собрать всех на одной странице, убедиться, что все общаются и следят за проектом.

Скажем так, большинство членов команды индивидуально мало вносят свой вклад в проект, и без руководства он так и остается. Но хороший руководитель проекта может объединить навыки членов команды так, чтобы производительность взлетела до небес.

Есть 3 способы управления командой:

Agile команда проекта ставит цели

Менеджер проекта должен сообщить участникам, над чем они работают. На основе задания вы должны определить цели команды, и все участники должны знать и поддерживать их. Цели команды должны быть разработаны таким образом, чтобы их можно было легко разбить на задачи. Задачи, которые в свою очередь соответствуют навыкам членов команды.

Наблюдая за командой

Наблюдение за командой проекта означает:

- Мониторинг производительности членов команды.
- Понять, что они делают и каков результат их работы.

Наблюдение помогает руководителю проекта отслеживать эффективность работы каждого члена команды, отслеживать ход выполнения проекта и выявлять проблемы на раннем этапе.

Управление конфликтами

В проектной команде, где вместе работают люди с разными характерами и стилями работы, неизбежно возникнет конфликт. У людей разные точки зрения, и в данных обстоятельствах эти различия перерастают в одну. конфликт. Роль менеджера проекта - разрешить этот конфликт.

Например, во время встречи по проекту вы просите всех членов команды оценить проект на данный момент. Потом выясняется, что члены команды не сотрудничатьони просто делают свое дело и не придерживаются соглашений. Как руководитель проекта, вы можете делать вещи 3: игнорировать конфликт, обвинять кого-либо или разрешать конфликт как можно быстрее. Вы можете догадаться, какое решение лучше.

Есть ряд действий, которые вы можете предпринять, чтобы разрешить этот конфликт:

- Проведите собрание команды, на котором члены команды должны признать наличие проблемы. Подтвердите (новые) соглашения.
- Позвольте членам команды понять важность сотрудничества в проекте.
- Попросите членов команды работать вместе и разрешить конфликт.

Самая важная вещь в процессе разрешения проблем - поддерживать общение открытым для всех. Участники должны поговорить о конфликте и обсудить свои чувства.

Конфликты можно обнаружить и устранить немедленно и быстро. Уважая различия между людьми, разрешая конфликты, когда они возникают, и опережая их, вы можете поддерживать здоровую и творческую командную атмосферу.

Что это значит для гибких проектных команд?

В среде Agile - Scrum распределение ролей совсем другое. Четкого менеджера проекта нет, и команда сама принимает важные решения. Поэтому команда находится в месте руководителя проекта и должна заниматься теми же вопросами.

- Команда должна будет принять задание проекта.
- Распределите обязанности команды проекта Agile.
- Команде придется договариваться о стратегии и методе.

- Члены команды должны будут мотивировать друг друга.
- Помещая правильные пользовательские истории в спринт, команда ставит свои собственные цели.
- Команда ежедневно отслеживает прогресс друг друга.
- Конфликты разрешаются внутри команды.

Agile Scrum есть хорошие инструменты для этого в виде встреч. Ежедневная встреча Stand-Up является хорошим примером этого. Каждый день настраивать, как висит флаг.

в 2001 году в штате Юта (США) на курорте Snowbird собрались семнадцать разработчиков программного обеспечения. В результате обсуждения методов разработки был опубликован «Манифест о гибкой разработке программного обеспечения Agile» (в переводе с английского понятие «agile» означает «подвижный», «проворный» или «быстрый», но в большинстве случаев его переводят именно как «гибкий»). Он и задал темп всей дальнейшей работе над созданием ПО.

Манифест Agile

Манифест, созданный программистами, включает в себя 4 базовых идеи и 12 принципов эффективного управления проектами. Любая из систем управления проектами на основе Эджайл (о системах мы поговорим позже) опирается именно на эти идеи и принципы, хотя и использует их в разных вариациях.

Идеи Agile:

1. Люди и их взаимодействие важнее, чем процессы и инструменты
2. Рабочее ПО важнее, чем документация
3. Клиенты и сотрудничество с ними важнее, чем контракт и обсуждение условий
4. Готовность к внесению изменений важнее, чем первоначальный план

Принципы Agile:

1. Удовлетворять клиентов, заблаговременно и постоянно поставляя ПО (клиенты довольны, когда рабочее ПО поступает к ним регулярно и через одинаковые промежутки времени)
2. Изменять требования к конечному продукту в течение всего цикла его разработки
3. Поставлять рабочее ПО как можно чаще (раз в неделю, в две недели, в месяц и т.д.)
4. Поддерживать сотрудничество между разработчиками и заказчиком в течение всего цикла разработки

5. Поддерживать и мотивировать всех, кто вовлечен в проект (если команда мотивирована, она намного лучше справляется со своими задачами, нежели команда, члены которой условиями труда недовольны)

6. Обеспечивать непосредственное взаимодействие между разработчиками (возможность прямого контакта способствует более успешной коммуникации)

7. Измерять прогресс только посредством рабочего ПО (клиенты должны получать только функциональное и рабочее программное обеспечение)

8. Поддерживать непрерывный темп работы (команда должна выработать оптимальную и поддерживаемую скорость работы)

9. Уделять внимание дизайну и техническим деталям (благодаря эффективным навыкам и хорошему дизайну команда проекта получает возможность постоянного совершенствования продукта и работы над его улучшением)

10. Стараться сделать рабочий процесс максимально простым, а ПО – простым и понятным

11. Позволять членам команды самостоятельно принимать решения (если разработчики могут сами принимать решения, самоорганизовываться и общаться с другими членами коллектива, обмениваясь с ними идеями, вероятность создания качественного продукта существенно возрастает)

12. Постоянно адаптироваться к меняющейся среде (благодаря этому конечный продукт будет более конкурентоспособен)

Треугольник ограничений проекта

Управление командой проекта

В спиральной модели удалось более четко и естественно определить контрольные точки проекта, в определенной степени, подчеркнув эволюционную природу жизненного цикла

Модель производственной архитектуры

✓ ориентирован на динамическое формирование требований; ✓ обеспечение их реализации; ✓ постоянного взаимодействия внутри самоорганизующихся рабочих групп; группы состоят из специалистов различного профиля.

Основные задачи: 1) Подбор и управление командой. 2) Выбор процесса. 3) Выбор инструментальных средств. ***Основные вопросы управления командой проекта:*** – ролевая модель команды; – модели организации команд; – общение в команде.

Модель производственной архитектуры

Управление программным проектом включает решение трех основных задач:

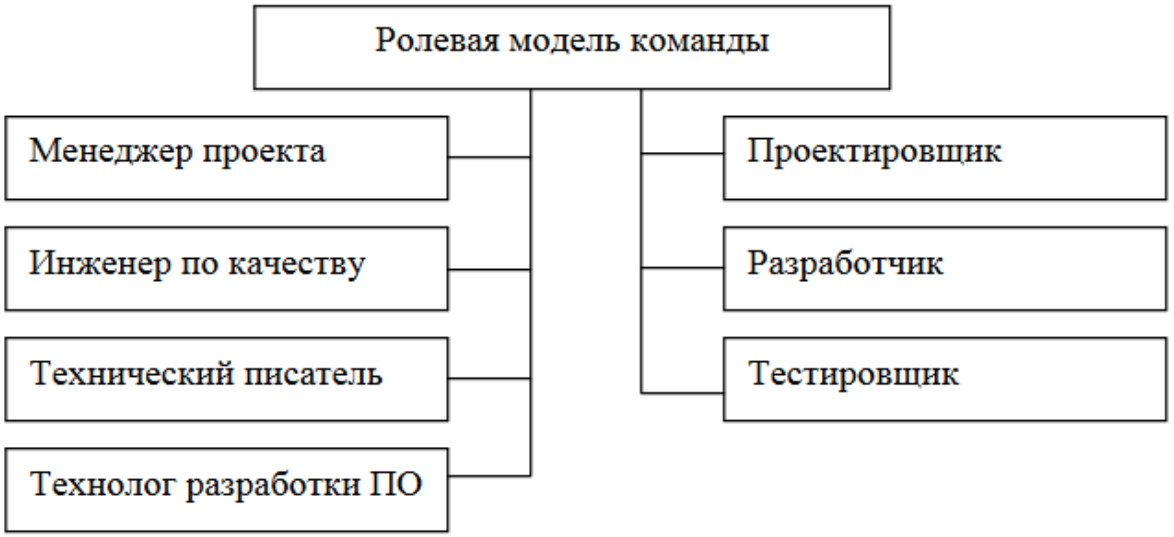
1. Подбор и управление командой (персонала).
2. Выбор процесса.
3. Выбор инструментальных средств.

Из множества вопросов управления командой проекта мы рассмотрим три:

- Ролевая модель команды;
- Модели организации команд;
- Общение в команде.

Ролевая модель команды

Состав команды определяется опытом и уровнем коллектива, особенностями проекта, применяемыми технологиями и уровнем этих технологий. На рисунке 4.1 представлен один из вариантов состава команды, описанный в [34]. Выделенные позиции не обязательно представлены конкретными людьми. Это список основных функциональных ролей в команде (ролевая модель команды). В малых командах роли могут совмещаться. В больших – выделяться группы или отделы (отдел проектирования, отдел тестирования, отдел контроля качества, отдел подготовки документации, ...).
Состав команды определяется также типом выполняемых работ: под заказ или коробочное производство (продукт на рынок). Инженерный психолог и инженер по маркетингу нужны в последнем случае. В представленной модели выделены следующие основные роли (рисунок 4.2):



- Менеджер проекта - главное действующее лицо:
  - Подбор и управление кадрами;
  - Подготовка и исполнение плана проекта;
  - Руководство командой;
  - Обеспечение связи между подразделениями;
  - Обеспечение готовности продукта.
- Проектировщик - это функция проектирования архитектуры высокого уровня и контроля ее выполнения. В небольших командах функция распределяется между менеджером и разработчиками. В больших проектах это может быть целый отдел.

⎪ Анализ требований.
⎪ Разработка архитектуры и основных интерфейсов.
⎪ Участие в планировании проекта.
⎪ Контроль выполнения проекта.
⎪ Участие в подборе кадров.

- Разработчик – роль, ответственная за непосредственное создание конечного продукта.

Помимо собственно программирования (кодирования) в его функции входит:

⎪ Контроль архитектурных и технических спецификаций продукта.

⎪ Подбор технологических инструментов и стандартов.
⎪ Диагностика и разрешение всех технических проблем.
⎪ Контроль за работой разработчиков документации, тестирования, технологов.
⎪ Мониторинг состояния продукта (ведение списка обнаруженных ошибок).
⎪ Подбор инструментов разработки, метрик и стандартов. Контроль их использования.

– Тестировщик – роль, ответственная за удовлетворение требований к продукту (функциональных и нефункциональных).

В функции тестировщика входит:

⎪ Составление плана тестирования. План тестирования составляет один из элементов проекта и составляется до начала реализации (разработки) проекта. Время, отводимое в плане на тестирование может быть сопоставимо с временем разработки.

⎪ Контроль выполнения плана. Важнейшая функция контроля – поддержка целостности базы данных зарегистрированных ошибок. В этой базе регистрируется:

– кто, когда и где обнаружил, описание ошибки, описание состояния среды;
– статус ошибки: приоритет, кто разрешает
– состояние ошибки: висит, в разработке, разрешена, проблемы

Эта база должна быть доступна всем, т.к. в тестировании принимают участие все члены команды.

⎪ Разработка тестов. Самая трудоемкая часть в работе тестировщика. Тестирование должно обеспечить полную проверку функциональности при всех режимах работы продукта.

⎪ Автоматизация тестирования включает автоматизацию составления тестов, автоматизацию пропуска тестов и автоматизацию обработки результатов тестирования. В виду важности автоматизации тестирования, иногда вводят нового участника – инженера по автоматизации.

⎪ Выбор инструментов, метрик, стандартов для организации процесса тестирования.

⎪ Организация Бета тестирования - тестирования почти готового продукта внешними тестерами (пользователями). Эту важную процедуру надо продумать и организовать в случае разработки коробочного продукта.

– Инженер по качеству. В современном представлении рассматривается три аспекта (уровня) качества:
  - –качество конечного продукта – обеспечивается тестированием,
  - –качество процесса разработки (тезис: для повышения качества продукта надо повысит качество процесса разработки),
  - –качество (уровень) организации (тезис: для повышения качества процесса надо повысить качество организации работ).

– Технический писатель или разработчик пользовательской (и иной) документации как части программного продукта. Функциями технического писателя являются:

- Разработка плана документирования, который включает состав, сроки подготовки и порядок тестирования документов.
- Выбор и разработка стандартов и шаблонов подготовки документов
- Выбор средств автоматизации документирования
- Разработка документации
- Организация тестирования документации.
- Участие в тестировании продукта.

Технический писатель все время работает с продуктом (его готовыми версиями) и выступая от имени пользователя видит все недочеты и несоответствия.

– Технолог разработки ПО обеспечивает выполнение следующих задач:
  - Поддержка модели ЖЦ - создание служб и структур по поддержке работоспособности принятой модели ЖЦ ПО. В поддержке модели ЖЦ принимают участие все. Но контроль возложен на технолога.
  - Создание и сопровождение среды сборки продукта. Функция особенно важна на завершающих этапах разработки или при использовании модели прототипирования. В такой ситуации сборка будет проводиться достаточно часто (в некоторых случаях - ежедневно). Среда сборки должна быть подготовлена заранее, сборка должна проводиться быстро и без сбоев. С учетом сборки версий это не простая задача.
  - Создание и сопровождение процедуры установки с тем, чтобы каждая сборка устанавливалась автоматически с учетом версии и конфигураций сред.
  - Управление исходными текстами - сопровождение и администрирование системы управления версиями исходных текстов.

Подводя итог, следует отметить, что ролевые модели проектных команд могут быть самыми разнообразными.

Модели организации команд

«...методологи разрабатывают сложные системы, в которых есть весьма изменчивые и нелинейные компоненты – люди» Алистэр Коуберн
Проблемы человеческого фактора: – все разные – по характеру, темпераменту, активности, целям; – все похожие – участие в проекте объединяет людей общностью целей, поиском путей достижения этих целей. – различаются по типу: • индивидуалисты – члены команды; • генераторы идей – исполнители; • ответственные – безответственные; – постоянны и изменчивы; – многообразны.

Как организовать работу команды? Команды из 10 человек и команды из 500 человек? Есть ли различия и в чем они состоят? Надо ли организовывать работу по жесткой технологии или надо предоставить свободу действий? Можно ли найти методологию (технологию) выполнения проекта, обеспечивающую успех?

Алистер Коубен [35] – специалист в области технологий выполнения ИТ проектов - приводит данные 23 проектов различной степени сложности, выполнявшихся по различным технологиям и имеющие различные результаты. Пытаясь проанализировать результаты применения различных технологий в тех или иных условиях, он приходит к выводу:

- Практически любую методологию можно с успехом применять в каком-нибудь проекте.
- Любая методология может привести к провалу проекта.

Главную причину он видит в том, прямо перед нами всегда находится нечто, чего мы не замечаем: люди. Именно человеческие качества обеспечивают успех тому или иному проекту, именно они являются фактором первостепенной важности, основываясь на котором надо строить прогнозы о проекте.

Исследованию вопросов человеческого фактора (Peopleware) уделяется достаточно много внимания, наиболее известными работами являются [36-39].

Проблемы человеческого фактора связаны с тем (проявляются в том), что участвующие в проекте люди:

- Все разные – по характеру, темпераменту, активности, целям – нет двух одинаковых людей.
- Все похожие – участие в проекте объединяет людей общностью целей, поиском путей достижения этих целей.
- Различаются по типу:
    - индивидуалисты - члены команды;
    - генераторы идей - исполнители;
    - ответственные – безответственные;
- Постоянны и изменчивы – люди, как правило, проявляют постоянство своих привычек и свойств характера, но при этом способны проявлять «противоположные» качества: индивидуалист – командные качества, исполнитель – генерировать идеи,
- Многообразны – надо понимать, что многообразие людей является основной гарантией выживания человечества вообще и возможности выполнять ИТ проекты в частности. Если бы все были индивидуалисты или все командники, все генераторы идей или все исполнители, то вряд ли удалось выполнить хотя бы один проект, а мир стал бы ужасен.

Как же управлять такими людьми? Рассмотрим основные модели проектных групп, которые применяются на практике

Административная модель (теория X)

Характерные черты модели:
– властная пирамида – решения принимаются сверху-вниз;
– четкое распределение ролей и обязанностей;
– четкое распределение ответственности;
– следование инструкциям, процедурам, технологиям;
– роль менеджера: планирование, контроль, принятие основных решений.

Это традиционный стиль управления, связанный с иерархической административно-командной моделью, которую используют военные организации. В основе лежит теория X, которая утверждает, что такой подход необходим, поскольку большинство людей по своей природе не любит работу и будет стремиться избежать ее, если у них есть такая возможность. Однако менеджеры должны принуждать, контролировать, направлять сотрудников и угрожать им, чтобы получить от них максимальную отдачу. Девиз теории и модели: Люди делают только то, что вы контролируете. Или в более мягком варианте: Люди делают то, что они не хотят делать, только если вы их контролируете В конце концов, теория утверждает, что большинство людей предпочитают, чтобы им говорили, что следует делать и им не придется ничего решать самим.
Характерные черты модели:
– Властная пирамида – решения принимаются сверху-вниз;
– Четкое распределение ролей и обязанностей;
– Четкое распределение ответственности;
– Следование инструкциям, процедурам, технологиям;
– Роль менеджера: планирование, контроль, принятие основных решений.
Преимущества модели: ясность, простота, прогнозируемость. Модель хорошо очетается с каскадной моделью жизненного цикла и применима в тех же случаях, что и каскадная модель. Модель эффективна в случае установившегося процесса.
Недостатки модели связаны с тем, что административная система стремится самосохранению (стабильности) и плохо восприимчива к изменению ситуации – новые типы проектов, применение новых технологий, оперативная реакция на изменение рынка.
Кроме того, в административной модели плохо уживаются индивидуалисты и генераторы идей.
Административная система (модель) – это тяжелый паровоз, идущий в «середине» и не поддающийся на «крайности» поиска новых путей и решений. Она воспринимает новые решения и технологии, но только проверенные, отработанные и стандартизированные. В этом ее сила, слабость и проявление принципа многообразия. Видимо, именно к ней в наибольшей степени применим термин «промышленное программирование».

Модель хаоса (теория Y)

Характерные черты модели:

– отсутствие явно выраженных признаков власти;
– роль менеджера – поставить задачу, обеспечить ресурсами, не мешать и следить, чтобы не мешали другие;
– отсутствие инструкций и регламентированных процедур;
– индивидуальная инициатива – решения по проблеме принимается там, где проблема обнаружена;
– процесс напоминает творческую игру участников на основе дружеской соревновательности.

В основе модели хаоса лежит Теория Y, которая является полной противоположностью Теории X. Основной тезис: Работа — естественная и приятная деятельность и большинство людей, на самом деле, очень ответственны и не увиливают от работы.
Характерными чертами модели хаоса являются:
– Отсутствие явно выраженных признаков власти;
– Роль менеджера – поставить задачу, обеспечить ресурсами, не мешать и следить, чтобы не мешали другие;
– Отсутствие инструкций и регламентированных процедур;
– Индивидуальная инициатива - решения по проблеме принимается там, где проблема обнаружена;
– Процесс напоминает творческую игру участников на основе дружеской соревновательности.
Преимущества такой модели в том, что творческая инициатива участников ничем не связана и потенциал участников раскрывается в полной мере. Это бывает особенно эффективно в случае, когда для решения проблемы требуется поиск новых подходов, методов, идей и средств. Команда становится командой «прорыва», а работа проходит в форме игры, цель которой – поиск наилучшего результата. Процесс напоминает случайный поиск, когда идеи и решения рождаются при живом и как бы случайном обсуждении проблем в коридоре, столовой на пикнике. Собрать такую команду в рабочей комнате и устроить обсуждение по регламенту часто просто не удается – это команда творческих индивидуалистов. Недостатки модели связаны с тем, что при определенных условиях команда
прорыва может стать командой провала. Причинами провала могут быть:
– Творческая соревновательность переходит в конкуренцию сначала идей, а потом - личностей.
– Процесс начинает преобладать над целью проекта – высказанные идеи не доводятся до конца и сменяются новыми идеями, преобладание получают «красивые» идеи, лежащие в стороне от основных целей проекта.
– Люди, способные к генерации идей, редко обладают терпением доведения идей до полной реализации.
Модель хаоса – это то, что нужно для освоения новых земель. Модель хаоса не противоречит административной модели – она ее дополняет и может эффективно с ней соседствовать (но в разных комнатах!). Многие мускулистые корпоративные бегемоты полагаются на исследовательские «отделы скунсов», откуда они черпают новые идеи, технологии и продукты.

Открытая архитектура (теория Z)

Характерные черты модели:
– адаптация к условиям работы – если делаем независимые модули, то расходимся и делаем, если нужна архитектура базы данных, то собираемся вместе и обсуждаем идеи;
– коллективное обсуждение проблем, выработка консенсуса и принятие решения, обязательного для всех;
– распределенная ответственность;
– динамическое изменение состава рабочих групп в зависимости от текущих задач;
– отсутствие специализации;
– задача менеджера – активное участие в процессе, контроль обсуждений, обеспечение возможности активного участия всех.

Административная и хаотическая модели являются двумя «крайностями», между которыми находятся множество моделей, сочетающих преимущества «крайних» моделей. Одной из таких моделей является модель открытой архитектуры, основанная на Теории Z. Эта теория была сформулирована Уильямом Оучи на основе изучения опыта японского стиля управления (Theory Z: How American Business Can Meet the Japanese Challenge,» Perseus Publishing, 1981). Теория Z предполагает (но не декларирует) наличие внутреннего механизма управления, основанного на влиянии со стороны коллег и группы в целом. Дополнительное воздействие оказывают культурные нормы конкретной корпорации.
Основной принцип модели можно сформулировать так: «Работаем спокойно. Работаем вместе».
Особенностями этой модели являются:
– Адаптация к условиям работы – если делаем независимые модули, то расходимся и делаем, если нужна архитектура базы данных, то собираемся вместе и обсуждаем идеи.
– Коллективное обсуждение проблем, выработка консенсуса и принятие решения – не все могут согласится, но принятое решение является коллективным и в силу этого – обязательным для всех.
– Распределенная ответственность – отвечают все, кто обсуждал, вырабатывал, принимал.
– Динамика состава рабочих групп в зависимости от текущих задач.
– Отсутствие специализации – участники меняются ролями и функциями и могут при необходимости заменить друг друга.
– Задача менеджера – активное (но рядовое, не руководящее) участие в процессе, контроль конструктивности обсуждений, обеспечение возможности активного участия всех.

Открытая архитектура является более гибкой, адаптируемой, настраиваемой на ситуацию. Она дает возможность проявить себя всем членам команды – в ней могут

уживаться и индивидуалисты и коллективисты. Коллективное обсуждение высказанных идей позволяет оставлять только прагматичные идеи.

Модель проектной группы MSF for Agile Software Development

Характерные черты модели:

Основные принципы построения команды

Рассмотрим, наверное, самую главную из отличительных черт MSF в сравнении с другими методологиями - модель проектной группы и принципы, положенные в основу построения команды.

Методология MSF считает, что успешная работа команды над проектом существенным образом зависит от ее структуры и распределения зон ответственности ролевых групп (более подробно о составе проектной группы далее) внутри команды. Построение команды в MSF соответствует ряду ключевых концепций (key concepts), часть которых кажутся самоочевидными (первые три), другие чем-то сродни «ноу-хау» (последние две):

- Концентрация на нуждах заказчика (customer-focused mindset) - главный приоритет любой хорошо работающей проектной группы. Означает обязательное понимание бизнес-задач заказчика и стремление к их решению со стороны команды. Не менее важным является активное участие заказчика в проектировании решения и получение его отзывов в ходе процесса разработки.
- Нацеленность на конечный результат (product mindset) - каждый участник проектной группы должен рассматривать собственную работу в качестве самостоятельного проекта или же вклада в какой-либо больший проект. Установка на конечный продукт означает, что получению конечного результата проекта уделяется больше внимания, чем процессу его достижения. Из этого не следует, что сам процесс может быть плох или непродуман - просто он существует для получения конечной цели, а не ради себя самого.
- Установка на отсутствие дефектов (zero-defect mindset) - это стремление к высочайшему уровню качества. Она означает, что цель команды – выполнение своей работы с максимально возможным качеством, в идеале таким образом, что если от команды потребуют поставить результат завтра, она будет способна поставить что-то работающее. В успешной команде каждый сотрудник чувствует ответственность за качество продукта. Она не может быть делегирована одним членом команды другому или же от одной ролевой группы другой.
- «Проектная группа - команда равных» (teem of peers). Концепция означает равноправное положение каждой из ролей в команде. Чтобы достичь успеха в рамках команды равных, каждый из ее членов, независимо от роли, должен нести ответственность за качество продукта, понимать интересы заказчика и сущность

решаемой бизнес-задачи. В то же время, принятие решения методом консенсуса между ролями не тождественно принятию решения методом консенсуса между сотрудниками. Каждая ролевая группа требует определенной организационной иерархии для распределения работы и управления ее ресурсами.

- Стремление к самосовершенствованию (willingness to learn) – это приверженность идее неустанного саморазвития посредством накопления опыта и обмена знаниями. Оно позволяет членам проектной группы извлекать пользу из отрицательного опыта сделанных ошибок, равно как и воспроизводить успехи, используя проверенные методы работы других людей. По окончанию основных фаз проекта и по завершению проекта в целом предполагается проведение открытых обсуждений его состояния и доброжелательный, но объективный анализ.

К концепции команды равных в MSF тесно примыкает идея о том, что каждая ролевая группа имеет зону ответственности и защищает интересы заинтересованных лиц из этой зоны (более подробно об этом далее).

Модель проектной группы в MSF может масштабироваться в зависимости от числа участников.

Ролевые группы и роли

Методология MSF основана на постулате о качественных целях, достижение которых определяет успешность проекта. Эти цели обуславливают модель проектной руппы. В то время как за успех проекта ответственна вся команда, каждая из ее ролевых групп, определяемых моделью, ассоциирована с одной из целей и работает над ее достижением. MSF for Agile Software Development выделяет 7 ролевых групп (см. рисунок 4.2):

- Управление программой (program management);
- Управление продуктом (product management);
- Управление выпуском (release operations);
- Архитектура продукта (architecture);
- Разработка (development);
- Тестирование (test);
- Удовлетворение потребителя (user experience);

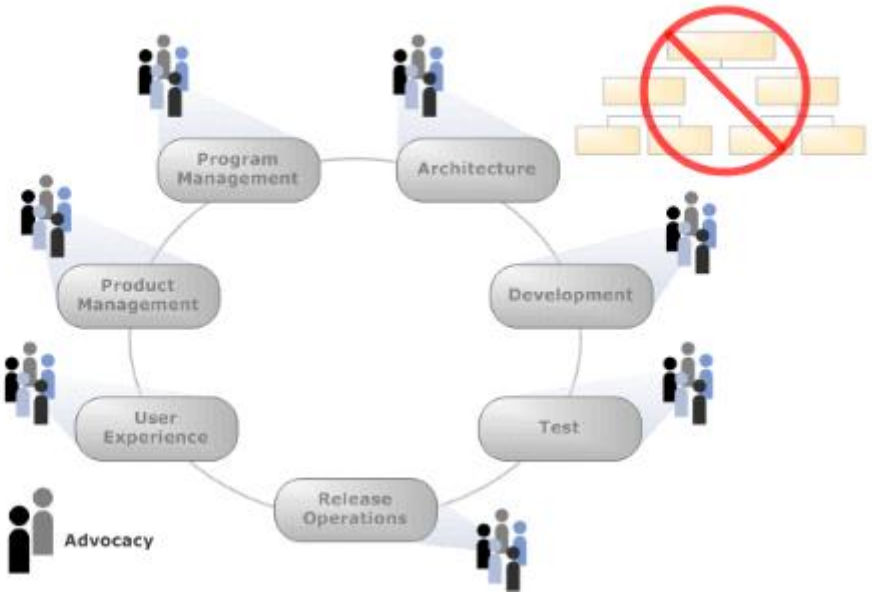


Рисунок 4.2 - Модель команды в MSF 4.0 – ролевые группы [36

и 6 ролей (см. рисунок 4.3):
¬ менеджер проекта (project manager) – ролевая группа «Управление программой», который отвечает за соблюдение ограничений проекта, его основная задача – вести процесс разработки таким образом, чтобы нужный продукт был выпущен в нужное время, он координирует деятельность всех членов группы, он должен полностью понимать все модели и процессы повышающие эффективность труда;

¬ бизнес-аналитик (business analyst) или менеджер продукта – ролевые группы «Управление продуктом» и «Удовлетворение потребителя», который отвечает за удовлетворение требований заказчика. Для этого менеджер продукта выступает представителем заказчика в группе разработчиков и представителем команды у заказчика. Менеджер продукта совместно с менеджером проекта должны прийти к компромиссному решению относительно функциональных возможностей продукта, сроков его разработки и финансирования проекта, так называемый треугольник ограничений проекта (рисунок 4.4). Существует закон Лермана:

«Любую техническую проблему можно преодолеть, имея достаточно времени и денег» (Следствие Лермана: «Вам никогда не будет хватать либо времени, либо денег»). Для того, чтобы решить проблему свойственной IT-проектам неопределенности и рискованности, одним из ключевых факторов их успеха являются эффективные компромиссные решения (trade-offs). Хорошо известна взаимозависимость между ресурсами проекта (людскими и финансовыми), его календарным графиком (временем) и реализуемыми возможностями (рамками). Эти три переменные образуют треугольник компромиссов (tradeoff triangle), приведенный на рисунке 3.17.

После достижения равновесия в этом треугольнике изменение на любой из его сторон для поддержания баланса требует модификаций на другой (двух других) сторонах и/или на изначально измененной стороне. Нахождение верного баланса между ресурсами, временем разработки и возможностями – ключевой момент в построении решения, должным образом отвечающего нуждам заказчика;

¬ релиз-менеджер (release manager) или логистик– ролевая группа «Управление выпуском», который отвечает за простоту развертывания и постоянное сопровождение. Логистик обязан разбираться в инфраструктуре продукта и требованиях к его сопровождению, кроме этого он должен уметь координировать установку программного обеспечения и оценить ее результаты. От него требуется хорошая техническая подготовка, он должен уметь устанавливать и настраивать

Рисунок 4.4 - Треугольник ограничений проекта

опыт развертывания крупномасштабных приложений на нескольких десятках или сотнях компьютерах. Одна из задач логистика – консультация группы сопровождения и пользователей после развертывания системы;

¬ архитектор (archrect) – ролевая группа «Архитектура», который отвечает за разработку дизайна проекта (продукта). Архитектор на самом деле входит в группу разработчиков, но на нем лежит ответственность за принятие проектных решений;

¬ разработчик (developer) – ролевая группа «Разработка», отвечает за реализацию системы и разрабатывает комплект проектной документации. Как программисты разработчики отвечают за низкоуровневое проектирование и оценку затрат на реализацию продукта. Разработчики, как правило, сами оценивают сроки выполнения своей работы. Такая концепция MSF – создание графиков ответственного за выполнение конкретного участка членами команды –называется составлением расписания «снизу-вверх» (рисунок 4.6). Такой подход повышает точность оценок (т.к. они основаны на конкретном опыте разработчика) и повышает ответственность исполнителя (т.к. план работ одновременно является обязательством члена команды). Планирование работ – задача всей проектной

группы, но в основании пирамиды лежит работа разработчиков. Менеджер программы добавляет в общий план работ некоторый резерв времени, чтобы гарантировать соблюдение сроков выполнения работ даже при возникновении непредвиденных обстоятельств.

¬ тестер (tester) – ролевая группа «Тестирование», отвечает за выявление и устранение проблем (дефектов) не только на уровне кода продукта, но и его спецификации и документацию к нему. При этом испытание продукта происходит в реальных условиях.

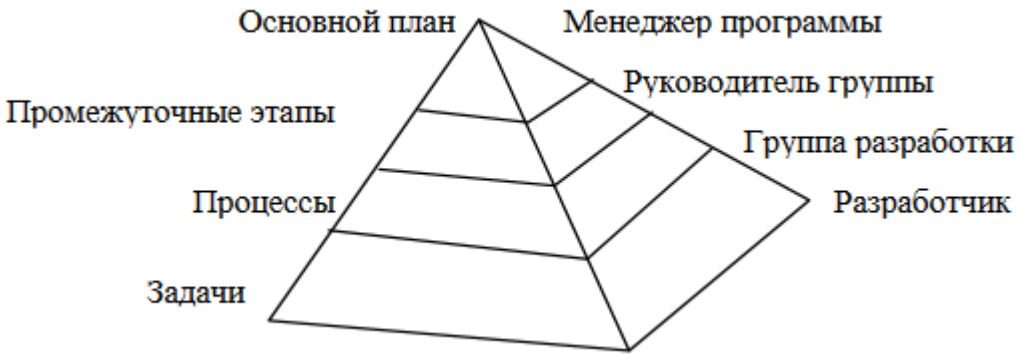


Рисунок 4.6 – Составление графика «снизу-вверх»

4.2.4.3 Рекомендации по возможному объединению ролей

Модель проектной группы в MSF может масштабироваться в зависимости от числа участников. Масштабирование модели проектной группы MSF for Agile Software Development на случай больших команд выходит за рамки нашего курса. Мы рассмотрим ситуацию, когда один человек может выполнять в проектной группе несколько ролей (небольшая команда, относительно несложный проект). В этом случае MSF предлагает рекомендации по возможному объединению ролей (таблица 4.1).

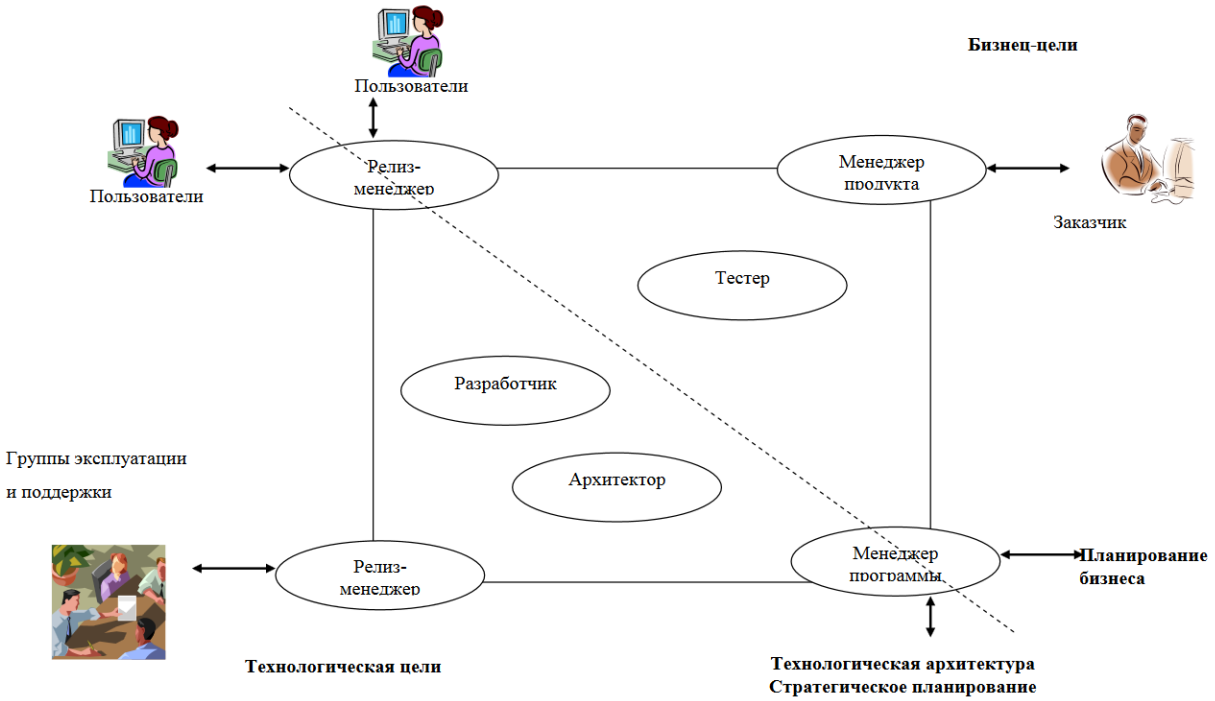
Таблица 4.1 – Возможное объединение ролей

	Архитектура продукта	Управление продуктом	Управление программой	Разработка	Тестирование	Удовлетворение потребителя	Управление выпуском
Архитектура продукта		Нет	Да	Да	Не желательно	Не желательно	Не желательно
Управление продуктом	Нет		Нет	Нет	Да	Да	Не желательно
Управление программой	Да	Нет		Нет	Не желательно	Не желательно	Да
Разработка	Да	Нет	Нет		Нет	Нет	Нет
Тестирование	Не желательно	Да	Не желательно	Нет		Да	Да
Удовлетворение потребителя	Не желательно	Да	Не желательно	Нет	Да		Не желательно
Управление выпуском	Не желательно	Не желательно	Да	Нет	Да	Не желательно	

4.2.4.4 Координация работы с внешними группами

Проектная группа, рассчитывающая на успех, должна взаимодействовать с внешними группами – как с заказчиками, так и с пользователями, так и с другими разработчиками [40]. Этим занимаются менеджер программы, менеджер продукта (бизнес-аналитик) и релиз- менеджер (логистик). В обязанность этих ролей входят как внутренние, так и внешние контакты, в то время, как архитекторы, разработчики и тестеры изолированы от общения с внешним миром (такая изоляция приводит к повышению эффективности труда этих двух групп). Важно, чтобы взаимодействие этих групп было понятным и явным. На рисунке 4.7 проиллюстрирована координация взаимодействий, связанная как с бизнесом, так и с технологиями.

Рисунок 4.7 – Взаимодействие проектной группы с представителями заказчика



Учебный пример. Формирование команды

Пусть проектная группа состоит из 4 человек. Представим возможное распределение ролей, позволяющее выполнить поставленную учебную задачу.

Каждому участнику нашей команды невозможно выдать по одной роли.

Разработчиков должно быть больше одного, а остальные роли придется совмещать.

Как именно, сейчас обсудим.

Во-первых, следуя рекомендациям MSF по объединению ролей, дадим одному из разработчиков еще и роль архитектора.

Во-вторых, отбросим в сторону другую крайность – разработчиками не могут быть все. Отдельный участник команды должен заниматься тестированием. Ему же можно выдать «в нагрузку» роль бизнес-аналитика.

Незадействованными остались ролевые группы «Управление программой» и «Управление выпуском». Соответственно роли менеджер проекта и релиз-менеджер достаются еще одному участнику.

В итоге получаем следующее (возможное) распределение:

- Участник 1 – менеджер проекта и релиз-менеджер;
- Участник 2 – архитектор и разработчик;

– Участник 3 – бизнес-аналитик и тестер;
– Участник 4 – разработчик.

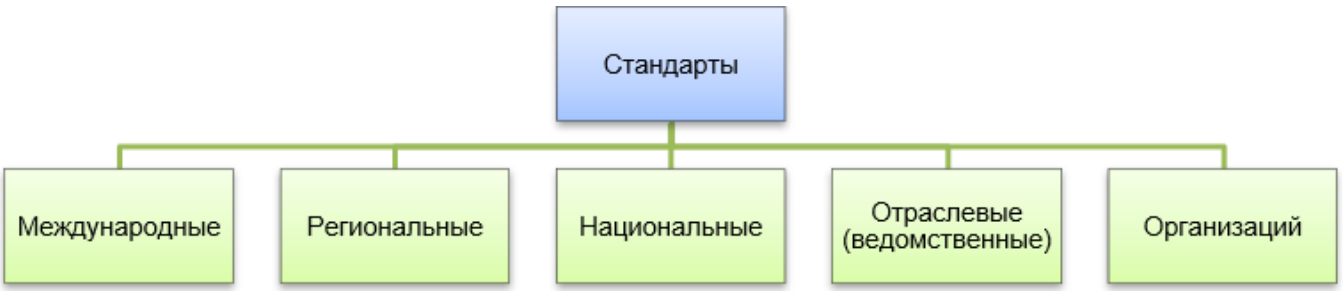
2.Стандарты программной инженерии

Стандарт (standard) – норма, образец, мерило
✓ утверждаемый компетентным органом нормативно-технический документ, устанавливающий комплекс норм и правил по отношению к объекту стандартизации, ✓ типовой образец, эталон, модель, принимаемые за исходные для сопоставления с ними других предметов.

Стандарт – исходный образец, эталон, модель для сопоставления с ним других подобных объектов.

Стандартизация:

- принятие соглашения по спецификации, производству и использованию аппаратных и программных средств вычислительной техники;
- установление и применение стандартов, норм, правил и т.п.



Основные типы стандартов

Международные стандарты разрабатываются специальными международными организациями на основе мирового опыта и лучших корпоративных стандартов. Имеют сугубо рекомендательный характер

Государственные стандарты (ГОСТы) принимаются государственными органами и имеют силу закона. Разрабатываются с учетом мирового опыта или на основе отраслевых стандартов. Могут иметь как рекомендательный, так и обязательный

характер. Для сертификации создаются государственные или лицензированные органы сертификации

Отраслевые стандарты действуют в пределах организаций некоторой отрасли (министерства). Разрабатываются с учетом требований мирового опыта и специфики отрасли. Являются, как правило, обязательными для отрасли. Подлежат сертификации

Корпоративные стандарты разрабатываются крупными фирмами с целью повышения качества своей продукции. Создаются на основе собственного опыта компании, но с учетом требований мировых стандартов. Не сертифицируются, но являются обязательными для применения внутри корпорации

Высший уровень стандартизации: международные организации по стандартизации.

<table>
<tr><th>Категория стандарта</th><th>Организация, принимающая (утверждающая) документы по стандартизации</th><th colspan="2">Обозначения документов по стандартизации</th></tr>
<tr><td rowspan="3">Международный</td><td>ISO – International Organization for Standardization (Международная организация по стандартизации, ИСО)</td><td>ISO</td><td rowspan="2">ISO/IEC</td></tr>
<tr><td>IEC – International Electrotechnical Commission (Международная электротехническая комиссия, МЭК)</td><td>IEC</td></tr>
<tr><td>ITU – International Telecommunication Union (Международный союз электросвязи, МСЭ)</td><td colspan="2">ITU-T Recommendation E, G, H, T, Q, X – рекомендации Сектора стандартизации электросвязи ITU</td></tr>
</table>

Разработчики стандартов в области программной инженерии

ISO – The International Standards Organization международная организация по стандартизации, работающая в сотрудничестве с IEC – The International Electrotechnical Commission – международной электротехнической комиссией. Наиболее представительная и влиятельная организация, разрабатывающая стандарты почти во всех областях деятельности, в том числе и в IT.

IEEE Computer Society – профессиональное объединение специалистов в области программной инженерии. Институт инженеров по электронике. Поддержка научных и практических разработок в области электроники и вычислительной техники. Большие вложения в разработку стандартов в этой области

ACM – Association for Computing Machinery – Ассоциация по вычислительной технике. Всемирная научная и образовательная организация в области вычислительной технике. Известна также и разработкой образовательных стандартов.

SEI – Software Engineering Institute – Институт Программной Инженерии при университете КарнегиМелон. Исследования в области программной инженерии с упором на разработку методов оценки и повышения качества ПО. Стандарты по качеству ПО и зрелости организаций, разрабатывающих ПО.

PMI – Project Management Institute – Международный Институт Проектного Менеджмента. Некоммерческая организация, целью которой является продвижение, пропаганда, развитие проектного менеджмента в разных странах. PMI разрабатывает стандарты проектного менеджмента, занимается повышением квалификации специалистов.

Объекты стандартизации в программной инженерии

- ✓ процессы разработки ПО
- ✓ продукты разработки
- ✓ ресурсы, которые используют процессы для создания программного продукта.

Основные стандарты программной инженерии

- ✓ ***ISO/IEC 12207*** – Information Technology – Software Life Cycle Processes - Процессы жизненного цикла программных средств.
- ✓ ***SEI CMM*** – Capability Maturity Model (for Software) – модель зрелости процессов разработки программного обеспечения.
- ✓ ***ISO/IEC 15504*** – Software Process Assessment – Оценка и аттестация зрелости процессов создания и сопровождения ПО. Является развитием и уточнением ISO 12207 и SEI CMM.
- ✓ ***PMBOK*** – Project Management Body of Knowledge – Свод знаний по управлению проектами.
- ✓ ***SWEBOK*** – Software Engineering Body of Knowledge – Свод знаний по программной инженерии.
- ✓ ***ACM/IEEE CC2001*** – Computing Curricula 2001 – Академический образовательный стандарт в области компьютерных наук.

Основные стандарты документирования приведены в лекции 16.

Ядро профессиональных знаний SWEBOK (Software Engineering Body of Knowledge)

Software Requirements – требования к ПО
Software Design – проектирование ПО
Software Construction – конструирование ПО
Software Testing – тестирование ПО
Software Maintenance – сопровождение ПО
Software Configuration Management – управление конфигурацией
Software Engineering Management – управление IT проектом
Software Engineering Process – процесс программной инженерии
Software Engineering Tools and Methods – методы и инструменты
Software Quality – качество ПО

Свод знаний по управлению проектами PMI PMBOK (Project Management Body of Knowledge)

Управление интеграцией – Project Integration Management
Управление содержанием – Project Scope Management
Управление временем – Project Time Management
Управление затратами – Project Cost Management
Управление рисками – Project Risk Management
Управление персоналом – Project Personnel Management
Управление коммуникациями – Project Communication Management
Управление закупками – Project Procurement Management
Управление качеством – Project Quality Management