ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
Федеральное государственное образовательное бюджетное учреждение
высшего образования
«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Я.А. Мостовой

УПРАВЛЕНИЕ ПРОГРАММНЫМИ ПРОЕКТАМИ

Учебное пособие

Самара
2016

УДК 004.41
ББК 32.973-018.2
М

Рекомендовано к изданию методическим советом ПГУТИ, протокол № 34, от 19.05.16.

Рецензенты:
Начальник отдела АО РКЦ ПРОГРЕСС, д.т.н., профессор Мантуров А.И.
Зав. кафедрой ИВТ ПГУТИ, д.т.н., профессор Бахарева Н.Ф.

Мостовой Я.А.
Управление программными проектами: учебное пособие / Я.А. Мостовой. – Самара: ПГУТИ. 2016. – 120 с.

Рассматривается управление проектной деятельностью в частности управление разработкой программного обеспечения. Пособие состоит из предисловия и 9 глав, охватывающих вопросы планирования, организации разработки, контроля и мотивации. Рассматриваются также вопросы управления рисками, роли руководителя, методы достижения компромисса и консенсуса при разработке программных проектов. В пособии приведены контрольные вопросы для самопроверки, а также материалы для практических занятий.

Для студентов направлений подготовки бакалавров: 27.03.04 «Управление в технических системах»; 09.03.01 «Информатика и вычислительная техника»; 09.03.04 «Программная инженерия»; 02.03.03 «Математическое обеспечение и администрирование информационных систем»

ISBN

ОГЛАВЛЕНИЕ

Системный подход к разработке ПО. Каскадная модель жизнен ного цикла ПО

Спиральная модель ЖЦ ПО.

Управление изменениями программного проекта

Тяжелые и легкие технологии разработки ПО.

SW-CMM (Capability Maturity Model for Software)

Модель компетентного разработчика PSP (Personal Software Process)

Глава 5. Декомпозиция (разбиение) СТС и ПО на подсистемы – универсальный метод снижения сложности разработки. Аутсорсинг. Организация разработки в большом. Организационная структура компании разработчика ПО.

Декомпозиция и аутсорсинг

Организация разработки ПО. Организация разработки в боль шом

Факторы успеха проекта**.** Норма управляемости. Управление проектами.

Норма управляемости. Проект и организационная структура компании

Функциональная форма структуры организации

Проектная форма структуры организации

Матричная форма структуры организации

Глава 6. Планирование разработки ПО. Сроки разработки

Зачем надо планировать разработку ПО

Задачи планирования

Планирование от трудоемкости разработки. Сетевые графики и их топология

Сроки графика разработки ПО и вопросы их коррекции

Диаграммы Ганта.

Глава 7. Управление рисками программного проекта. Методы контроля хода исполнения программных проектов.

Риски - нештатные ситуации процесса разработки

Отчего возникают риски
Планирование управления рисками. Идентификация рисков. Допущения проекта. Методы реагирования на риски
Наиболее распространенные риски программных проектов
Характеристики процессов контроля, принципы контроля
Как проверять планы и как оценивать ход их исполнения
Метрики проекта и техника его контроля

Глава 8. Управление проектом и лидерство. Работа руководителя.

Лидерство
Компетенции эффективного руководителя.
Стратегии руководства
Управление персоналом, мотивация
Конфликт и управление проектом в этих условиях

Глава 9. Достижение компромисса и консенсуса. Обзор систем управления проектами

Коммуникации при управлении программными проектами
Принятие решений при разработке ПО. Достижение компро мисса и консенсуса
Как добиться консенсуса?
MS Excel. Хорошо подходит для недельного планирования и от четности.
MS Project 2002
Open Plan. Система планирования и контроля крупных проек тов
Primavera Project Planner. (P3) применяется для управления средними и крупными проектами
SureTrak Project Manager. Продукт ориентирован на контроль выполнения небольших проектов
Spider Project.
1С-Рарус: Управление проектами. Российская разработка на платформе бухгалтерской системы "1С:Предприятие"

ПРЕДИСЛОВИЕ

Программное обеспечение – сложный продукт и основная трудоёмкость его создания и использования приходится на разработку т.е. именно на то, что называют проектной деятельностью. Вопросы тиражирования, стоящие остро для других продуктов, создаваемых человеком, для программного обеспечения не существенны.

Программное обеспечение компьютерных технологий создаётся в большинстве случаев коллективно и только правильное управление программными проектами приводит их разработчиков к успеху. Правильное управление в свою очередь связано с планированием, организацией, мотивацией и контролем процессов разработки.

Управление программными проектами – область деятельности, в ходе которой определяются и достигаются четкие цели программного проекта при балансировании между требованиями к системе и ПО, объёмом работ, потребными ресурсами (такими как деньги, труд, материалы, энергия, пространство и др.), временем, качеством и рисками.

Если топология плана (операции разработки и связи между ними) – объективная часть плана разработки, то сроки выполнений операций плана носят более субъективный характер, что отражает объективное желание руководителей проекта завершить работу в требуемые жизнью сжатые сроки. Надо отметить, что планируются сроки создания ПО, которого еще нет. Какие сроки проведения работ надо поставить на каждую из сотен позиций графика? Как доказать себе и руководителю, который традиционно выжимает минимальные сроки, обоснованность ваших предложений?

Это один из ключевых вопросов планирования и управления проектной деятельностью в целом. Здесь на помощь приходят методы оценки размера ПО и трудоемкости его разработки до начала программирования, полученные с помощью статистики по ранее разработанным программам, например, метод функциональных точек, а также опыт руководителя.

Программные проекты подвержены воздействию множества недостаточно первоначально определённых, а также случайных факторов. Цели управления рисками проекта – снижение вероятности и значимости воздействия (ущерба) от неблагоприятных для программного проекта факторов и событий. Здесь уместна аналогия с

управлением сложными техническими системами, для которого обязательным условием является наличия заблаговременно определённого перечня нештатных ситуаций и продуманного управления выходом из них, минимизирующего ущерб. Управление рисками процессов разработки программного обеспечения является средством минимизации вероятного ущерба от нештатных ситуаций процессов разработки, приводящих к невыполнению требований по качеству, бюджету и срокам разработки.

Статистика разработки программных проектов такова:

- только 35 % проектов завершились в срок, не превысив запланированный бюджет, и реализовали все требуемые функции и возможности.

- 46 % проектов завершились с опозданием, расходы превысили запланированный бюджет, требуемые функции не были реализованы в полном объеме.

-19 % проектов полностью провалились и были аннулированы до завершения.

Основной причиной большинства провалов программных проектов является именно применение неадекватных методов управления его разработкой: отсутствие или частые изменения требований, отсутствие необходимых ресурсов людских, материальных, временных, ошибки в оценках трудоемкости и сроков работ, использование слишком новых, неопробованных и нестабильных технологий разработки.

Необходимо учитывать эти причины при управлении программными проектами и правильно управлять людьми. Успех, а равно и провал, проектов по производству ПО во многом лежат в области психологии и организации работ. Руководство и лидерство, конфликты и методы их разрешения, достижение компромиссов и консенсуса – вопросы, возникающие при управлении программными проектами и рассмотренные в настоящем издании.

Данное учебное пособие соответствует аналогичному по названию курсу, читаемому студентам направлений подготовки бакалавров: 27.03.04 «Управление в технических системах»; 09.03.01 «Информатика и вычислительная техника»; 09.03.04 «Программная инженерия»; 02.03.03 «Математическое обеспечение и администрирование информационных систем»

В пособии присутствуют обзорные материалы по процессам разработки ПО.

Глава 1. Проекты и управление ими. Управление проектами - наука и искусство.

Программные проекты. Участники проекта.

Проект – это временное мероприятие, предназначенное для создания уникальных продуктов, изделий, услуг или результатов. Управление проектами означает применение знаний, инструментов и методов управления к проектной деятельности для удовлетворения предъявляемых к проекту технических требований в рамках выставленных ограничений по срокам исполнения и бюджету. Из данных определений следует, что всем проектам присуще две важные характеристики.

1.Наличие дат начала и завершения (у каждого проекта есть начало и конец, этим проектная деятельность отличается от операционной, рутинной деятельности).

2.Результат каждого проекта – уникальный продукт или новая услуга. Это ещё одно отличие проектной деятельности от операционной, в процессе которой серийно производится определённая продукция или оказывается постоянная определённая услуга.

Вычислительные машины были изобретены, как устройства, способные облегчить и ускорить проведение сложных расчетов, которые без них требовали для своего проведения много времени. Теперь же в большинстве случаев на первый план выходит способность ЦВМ получать, передавать, хранить и обрабатывать огромные объемы информации, а также управлять сложными физическими процессами. В системах компьютерного управления автоматизированными и автоматическими системами резко выросли возможности получения не достижимого ранее качества управления.

Это наряду с возможностью сравнительно легкого внесения новых стратегий функционирования без переоснащения и перепроектирования всей системы путем изменения программного обеспечения встроенных в систему компьютеров предопределили бурный рост компьютерных технологий в области автоматизации и управления. Программное обеспечение компьютерных технологий –

сложный в разработке и трудоемкий продукт, создаётся в большинстве случаев коллективно и только правильное управление программными проектами приводит их к успеху[6,7,9].

Управление программными проектами – область деятельности, в ходе которой определяются и достигаются четкие цели программного проекта при балансировании между требованиями к системе и ПО, встроенных в систему вычислительных средств, объёмом работ, ресурсами (такими как деньги, труд, материалы, энергия, пространство и др.), временем, качеством и рисками.

Во многих случаях в проекте выделяют роли заказчика, исполнителя и потребителя (иногда инвестора или спонсора). Такие роли почти всегда есть для внешних проектов, выполняемых для сторонних организаций. Для внутренних проектов – проектов, реализуемых внутри организации, такое разделение ролей также желательно с целью повышения эффективности при разделении труда и для устранения конфликтов при формировании требований и приёмке результатов, а также для определения зон ответственности.

Заказчик– физическое или юридическое лицо, которое определяет цель и ограничения проекта и его финансирование. В качестве заказчика часто выступают инвесторы, осуществляющие вложение в проект собственных, заемных или привлечённых средств. Результаты проекта принадлежат инвестору. В том случае, когда заказчик не является инвестором он наделяется правами владения результатами проекта на период и в пределах полномочий, установленных договором и в соответствии с законодательством. Заказчик несёт ответственность за постановку целей и полезность результата для потребителя.

Исполнитель (разработчик) выполняет проект согласно согласованному с заказчиком требованиями и планом. Если заказчик и исполнитель находятся в разных организациях, то составляется договор на исполнение проекта. Этот договор ссылается на документ, содержащий согласованные требования к продукту и условия его приемки и эксплуатации. При изменении требований заказчика может быть подписано с ним дополнительное соглашение к договору в рамках ограничений суммарного бюджета проекта, оговоренных основным договором или с привлечением дополнительных средств.

Пользователи проекта используют разработанный в рамках проекта продукт. Пользователем проекта могут быть заказчики, ин-

весторы, а также другие физические или юридические лица. Пользователи используют разработанный продукт в соответствии с эксплуатационной документацией, предоставляемой разработчиком. Чаще всего разработчик берёт на себя обязательства сопровождения проекта в течении оговорённого срока. Этот срок устанавливается договором на разработку с заказчиком.

Отдельно следует сказать о руководителе проекта, который отвечает за результаты выполнения проекта и осуществляет управление выполнением проекта.

Наука и искусство управления проектами.

При управлении сложными процессами так же, как и сложными объектами необходим научный подход. Прежде всего участникам проекта необходимы знания в предметной области, знания объекта управления. Ранее мы рассматривали в качестве объекта управления сложные технические системы и объекты [3,4,]. А теперь объект управления – сам сложный процесс производства ПО.

Программное обеспечение –сложный в производстве продукт. Вопросы технологии производства ПО, конструирования ПО, соответствующего заданным требованиям и внешним и внутренним характеристикам качества, рассматриваются в инженерии ПО. Согласно документу SWEBOK 2004 (SWEBOK - Software Engineering Body of Knolidge),, выпущенному для определения необходимого объема знаний для разработки ПО, программная инженерия включает в себя 10 основных и 7 дополнительных областей знаний, на которых базируются процессы разработки ПО [12].

К основным областям знаний относятся следующие области:

1. Software requirements – программные требования.
2. Software design – программный дизайн (архитектура).
3. Software construction – конструирование программного обеспечения.
4. Software testing – тестирование ПО.
5. Software maintenance – эксплуатация (поддержка) программного

обеспечения.

6. Software configuration management – конфигурационное управление ПО.

7. Software engineering management – управление программной инженерией.

8. Software engineering process – процессы программной инженерии.

9. Software engineering tools and methods – инструменты и методы программной инженерии.

10. Software quality – качество программного обеспечения.

Все это необходимо знать и уметь применять, для того чтобы разрабатывать ПО.

В то же самое время при управлении сложными процессами часто возникают ситуации, когда невозможно принять научно обоснованное решение просто из-за нехватки информации или времени на её обработку. В этом случае помогает, то, что называется **искусством управления – способность принимать по интуиции правильные решения** в условиях нехватки информации и времени. Психологами доказано, что интуиция основана на знаниях и практическом опыте своем и чужом, а также на способности некоторых людей очень быстро в уме моделировать и прогнозировать ситуацию.

Искусство управления требуется на всех уровнях иерархии общества. На верхних уровнях управления – высока неопределенность возникающих ситуаций, не хватает информации, знаний, времени, необходимых кадров и организаторских способностей.

На нижнем исполнительском уровне у участников проекта также не хватает информации и времени, не видна целостная картина процесса разработки проекта, поэтому трудности воспринимаются, как дефекты деятельности руководства.

На среднем уровне управления проектом к упомянутой нехватки информации и времени добавляется другая основная проблема – правильная линия поведения «между двух огней»: давления сверху от руководителей верхнего уровня и давление снизу от исполнителей. Не все умеют правильно выбрать эту линию.

Невежество и отсутствие способностей в области управления (той самой интуиции) – виновник многих бед. Дилетант и неспособный человек тем охотнее берётся за деятельность, чем больший интервал времени отделяет его от проявления результата деятельности.

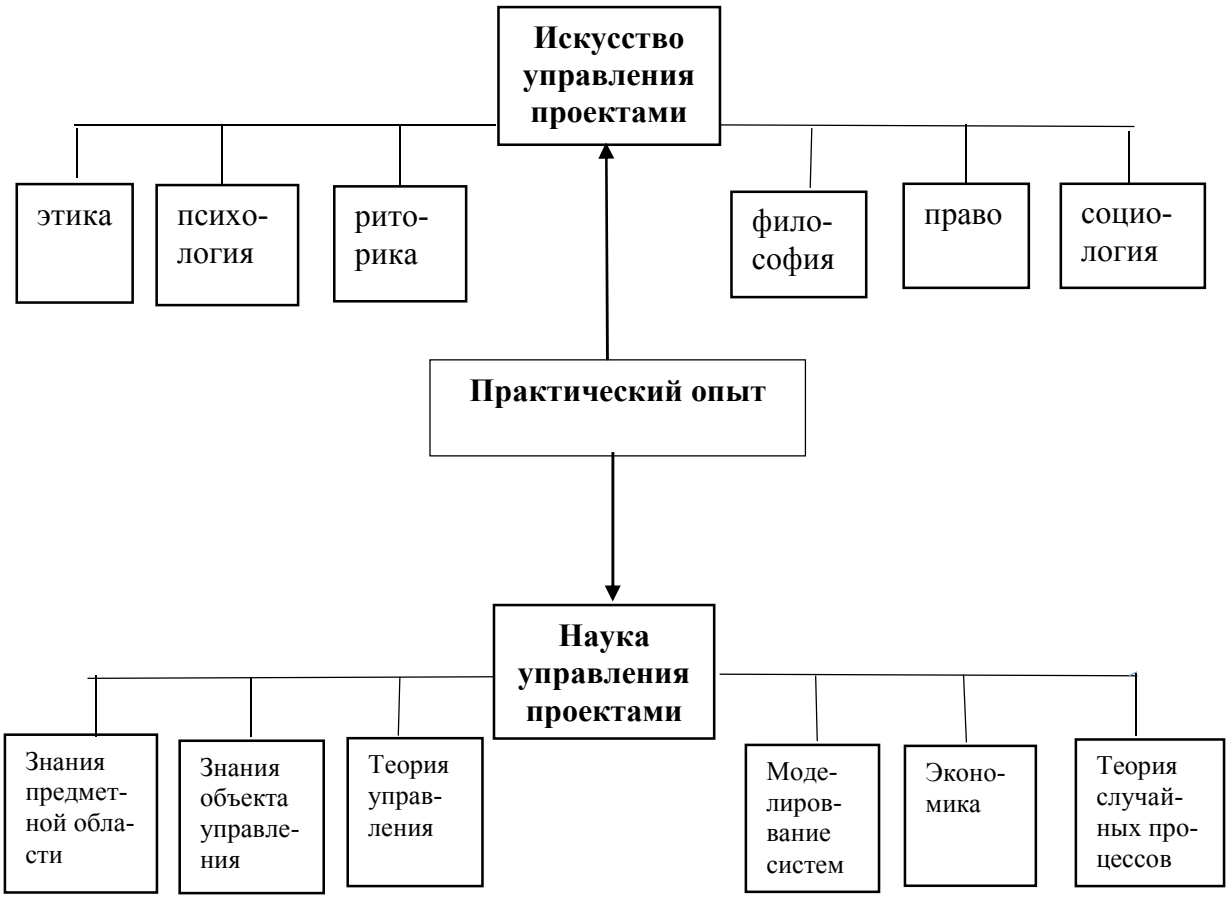


Рис. 1 Схема научных направлений и факторов, обеспечивающих УПП.

Для управления большими проектами и процессами в частности программными проектами этот интервал времени достаточно большой. Неумелое управление большим проектом, организацией начинает проявляться не сразу и ведет к разорению и негативным последствиям только через некоторое время, а пока плохо управляемый проект «катится по инерции». Поэтому многие люди охотно берутся за управленческие дела, рассуждая как Ходжа Насреддин, взявшийся научить осла эмира богословию за 10 лет (за 10 лет что - нибудь да случится либо с ослом, либо с эмиром, либо со мной). При этом никакой дилетант и неспособный человек не берётся за дела, которые сразу же дают очевидный и плачевный результат (сыграть Бетховена, решить математическую задачу). На рис. 1 приведены направления знаний, обеспечивающих управление проектами.

Операционная и проектная деятельность

Различают два вида рабочей человеческой деятельности: операционная и проектная. **Операционная деятельность** это деятельность, когда внешние условия хорошо известны и стабильны, когда производственные операции хорошо изучены и неоднократно испытаны, а функции исполнителей определены и постоянны. В этом случае основой эффективности деятельности служат узкая специализация и повышение квалификации исполнителей и управляющих. Нового в этом виде деятельности разрабатывать не надо.

Там, где разрабатывается новый продукт или услуга, внешние условия и требования к которому определены недостаточно полно и меняются, где применяемые производственные технологии используются впервые, где постоянно требуются поиск новых решений и возможностей, интеллектуальные усилия и творчество, где имеются риски получения отрицательных результатов там **имеет место проектная деятельность.**

Операционная деятельность и проектная различаются, главным образом, тем, что операционная деятельность – это продолжающийся во времени монотонный и повторяющийся рабочий процесс, в то время как проекты являются ограниченными по срокам и уникальными, имеющие задачу достижение конкретной новой бизнес-цели, например, разработка программного обеспечения. Задача операционной деятельности – обеспечение нормального течения бизнеса, например, эксплуатация уже разработанного программного обеспечения, работа системного администратора [1].

Ограничение проектов по срокам означает, что у любого проекта есть четкое начало и четкое завершение. Завершение наступает, когда достигнуты цели проекта; или осознано, что цели проекта не будут или не могут быть достигнуты; или исчезла необходимость в проекте. Если бы результаты проекта не носили уникальный творческий характер, работу по их получению можно было бы четко регламентировать, установив производственные нормативы, и реализовывать их, как бы, в рамках операционной деятельности. Попытки к такому положению дел предпринимаются постоянно, но проектная деятельность очень часто из-за непредвиденных обстоятельств нарушает все подобные планы.

Управление программными проектами. Главные причины провалов программных проектов

Управление проектами состоит из нескольких действий, начинающихся с определения целей. Далее следуют четыре действия собственно управления [8], состоящие из (см. рис. 2):

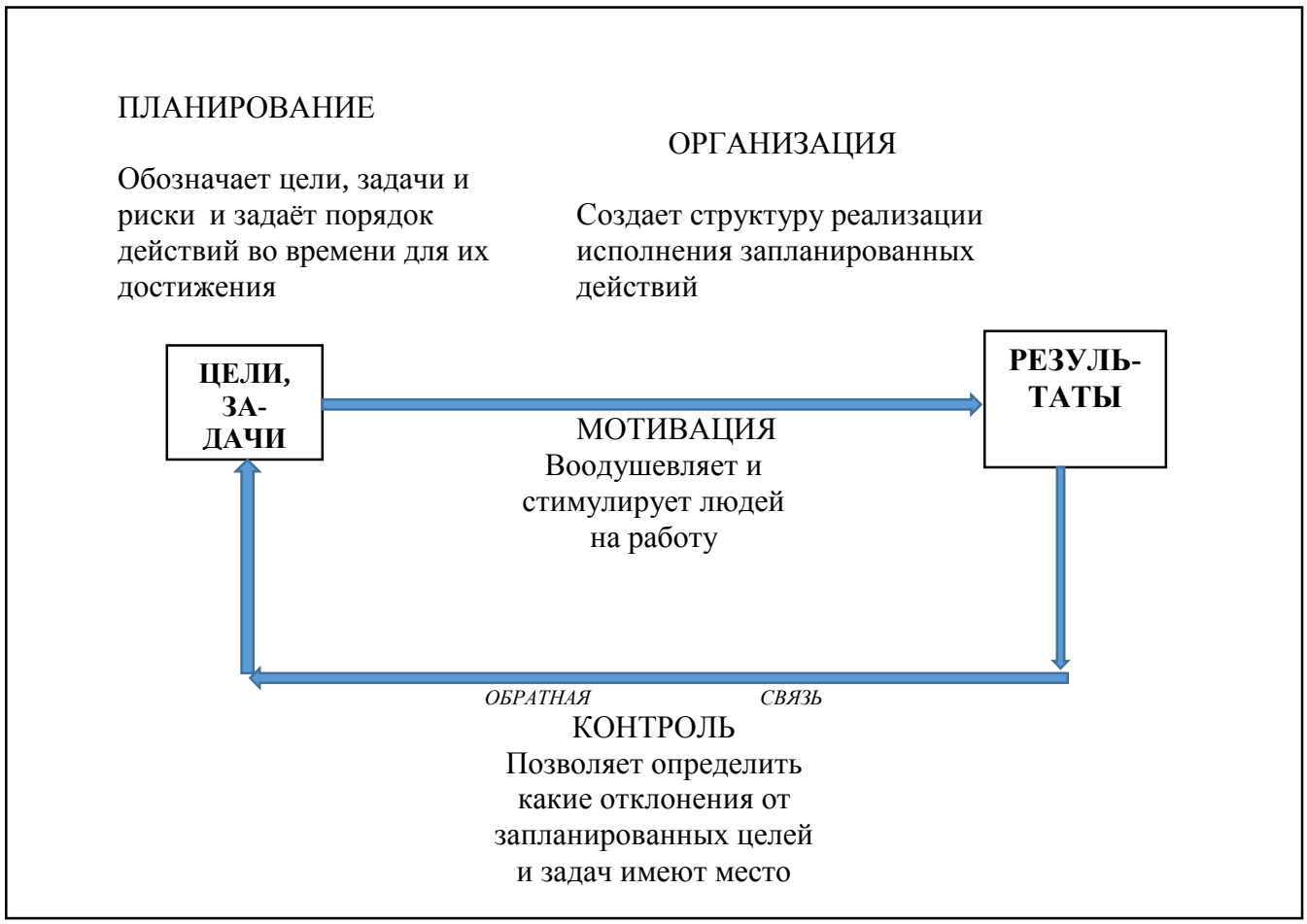


Рис.2 Основные операции управления проектом

- операций планирования – определения порядка действий для достижения целей и сроков выполнения каждого действия,

-создания структуры (организации) для управления проектом и реализации процессов его выполнения,

-мотивации разработчиков на успешное исполнение проекта,

- обратной связи управления через контроль выполнения запланированных работ.

То, что производят программисты – это коллективные (индивидуальные в случае индивидуальной разработки) мысли и идеи, выраженные на языке программирования. Творчество при разработке ПО начинается с определения целей программы и заканчивается

только тогда, когда в её коде, написанном на каком-либо языке программирования, поставлена последняя точка.

При этом задача непосредственно программирования – кодирования не является творческой, и большинство современных технологий разработки ПО направлены на упрощение и сокращение этого этапа рутинной работы [2,3,6].

Творческий элемент в программировании в основном связан с конструированием программ. В проблемно ориентированных языках программирования созданы средства, позволяющие фактически решать задачу на графоаналитическом, приближенном к естественному, языке предметной области. Конечно, этот язык только приближен к естественному, строго формализован и не допускает неоднозначностей. После завершения конструирования задачи запускается компилятор, который и даёт программный код.

Статистика разработки программных проектов такова:

- Только 35 % проектов завершились в срок, не превысив запланированный бюджет, и реализовали все требуемые функции и возможности.

- 46 % проектов завершились с опозданием, расходы превысили запланированный бюджет, требуемые функции не были реализованы в полном объеме. Среднее превышение сроков и затрат превышает 100%,

-19 % проектов полностью провалились и были аннулированы до завершения.

Основной причиной большинства провалов программных проектов является именно применение неадекватных методов управления его разработкой. Главными причинами провалов программных проектов являются:

1.Отсутствие требований заказчика или их неполнота, или их частые изменения.

2.Отсутствие рабочего взаимодействия с заказчиком.

3. Отсутствие опыта у разработчика.

4 Отсутствие необходимых ресурсов людских, материальных, временных.

5.Неполнота планирования, «забытые работы».

6. Ошибки в оценках трудоемкости и сроков работ.

7. Использование слишком новых, неопробованных и нестабильных технологий раз работки.

Что делать? Учитывать эти причины при управлении программными проектами и правильно управлять людьми. Успех, а равно и провал, проектов по производству ПО во многом лежат в области психологии и организации работ. Разрабатывать ПО точно не сложнее, чем разрабатывать, например, станки и ракеты.

Глава 2. Обзор метода функциональных точек. Размер ПО и связь его с трудоемкостью разработки

Задача оценки размера ПО

Размер ПО важен для некоторых приложений, где имеются ограничения на размер памяти системной ЦВМ. В этом случае мера размера ПО – число машинных команд программы, а не число строк исходного текста на ЯП.

Оценки размера ПО имеют также большое значение при планировании разработки ПО. Планирование разработки отталкивается от располагаемого числа исполнителей работ и от трудоемкости разработки, определяемой в человеко- днях. Если трудоемкость работ составляет 100 человеко-дней, и имеется 5 разработчиков, то эту работу они должны сделать за 20 рабочих дней. Но как узнать трудоемкость работы до начала её исполнения?

До сих пор трудоемкость разработки ПО связывается с размером ПО. Причем эта связь в простейших методиках (моделях трудоемкости разработки) представляется, как линейная.

$$\textbf{Трудоемкость} = \text{Число дней} \times \text{Число человек} = \frac{\textit{объем}}{\textit{числострок / день} \times \textit{человека}}$$

Здесь производительность труда программиста «число строк / день на человека» рассматривается, как константа, определяемая для той или иной технологии разработки ПО по статистике разработанных программ. В соответствии с данным выражением связь между трудоёмкостью и объёмом ПО представляется линейной. Действительно связь размера с трудоемкостью ПО имеется, но она далеко не линейна. И, например, при увеличении размера ПО в 2 раза, трудоемкость растет быстрее, чем в 2 раза (в 3-4 раза).

Не однозначен ответ на вопрос чем измерять объем ПО? Числом машинных команд? Числом строк исходного текста на языке

программирования? Числом функций, которое ПО выполняет? Все эти меры имеют право на жизнь и применяются, хотя исторически они формировались в приведенной последовательности.

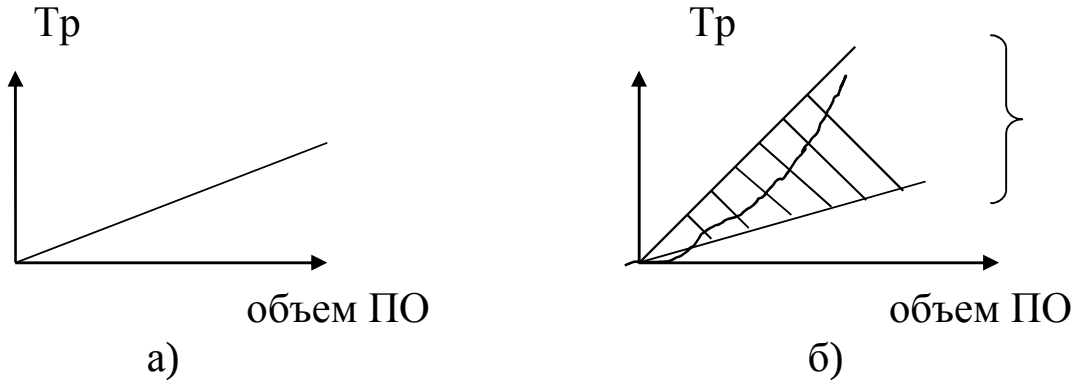


Рис.3 Простейшая связь между трудоёмкостью ПО и его объёмом – а), учёт нелинейности зависимости –б)

В случаях планирования разработки ПО и оценки его трудоемкости, современная мера размера ПО – число строк исходного текста на языке программирования (ЯП). Но и здесь нужно определиться: какого ЯП? Известно, что выразительность различных языков программирования различна и одна и та же функция, запрограммированная на различных ЯП, будет иметь различное число строк исходного текста.

Во всех случаях оценить размер ПО, когда оно создано, это простая задача. Взял и посчитал. А вот оценить ПО, когда оно не создано, но есть спецификация ПО – это практическая задача над которой бьётся много людей.

Конечно, для ответа на поставленный вопрос о размере ПО без его программирования необходимо наличие информации о **предполагаемых функциях ПО и об его устройстве.** Когда далее придётся говорить и о трудоемкости ПО**, то конечно нужна информация о технологической среде его разработки.**

Размер ПО и трудоемкость ПО и факторы, влияющие на них.

Альтернативой может служить функциональный показатель объема ПО не в строках исходного текста на ЯП, а в количестве так называемых функциональных точек, как меры функциональности оцениваемого ПО. Отметим, что функциональность ПО не зависит

от ЯП. Таким образом оценка размера ПО в числе ФТ не зависит от ЯП.

При этом функциональная точка включает в себя подпрограмму выполнения небольшой законченной функции в составе задачи, а так же некоторое ее программное обрамление, предназначенное для связи данной функции с программным окружением в котором эта функция существует. То есть с учетом ввода-вывода данных конкретного взаимодействия с пользователем, файлами из баз данных и т.д.

Оценка количества функциональных точек ПО возникает не на пустом месте и не берётся «с потолка». Размер ПО зависит от числа выполняемых ПО функций и их сложности, числа и видов данных обрабатываемых ПО. Вся эта информация содержится в концептуальной модели разрабатываемого ПО, которая известна до начала программирования и может быть составлена для подсчета количества ФТ.

Далее после определения размера ПО необходим оценка его трудоемкости для составления плана разработки. Известен факт, что две программы одного и того же размера могут иметь различную трудоемкость. Программы имеющие линейную структуру менее трудоемки и в проектировании, и в отладке, чем программы, имеющие сильно ветвящуюся и запутанную структуру. Это говорит о том, что приведенная на рис.3 линейная связь трудоемкости с размером ПО, конечно, не верна. Она не верна ещё и потому что с ростом размера программы растут трудозатраты на структуризацию – разбиению на части, создание и отладку связей, по которым происходит взаимодействие частей ПО между собой. В линейной модели «объем ПО - трудоемкость ПО» этому соответствует разброс псевдоконстанты «число строк / день*человек», который покрывает нелинейность рассматриваемой зависимости (см.рис.3).

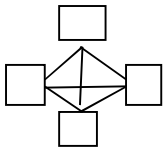
Поэтому ответы на поставленные нами вопросы, требуют введения раздельных методов и отдельных моделей как для размера ПО, так и для трудоемкости его разработки. Возможно учесть в моделях трудоемкости, базирующихся на размере ПО, конкретные для каждого ПО факторы среды разработки и исполнения, влияющих на трудоемкость, в виде дополнительных поправок к базовой зависимости «трудоемкость – объем ПО».

На производительность труда программистов «число строк / день*человек». оказывают влияние инструментальные средства и

технология разработки ПО. В среднем производительность труда программиста за последние 20 лет изменилась не значительно, не смотря на появление за это время множества новых ЯП.

В малых программных проектах разработка непосредственно кода является наиболее трудоемкой частью проекта. В больших проектах архитектура, структуризация, затем интеграция и комплексная отладка взаимодействия занимают все большее время, а доля создания непосредственно кода занимает все меньшее время. При этом можно считать, что трудозатраты на создание кода растут пропорционально его размеру, но затраты на другие виды деятельности в основном на организацию и проверку правильности взаимодействия растут в гораздо большей степени.

Так, если ПО разрабатывают два человека, то полное число связей по которому происходит взаимодействие между ними равно 1 , при взаимодействии 4 человек при разработке ПО полное число связей по которым идёт взаимодействие равно уже 6 (каждый с каждым), при 5 –равно10, при10 равно уже 45 (см.рис.4). С ростом размера кода и соответственно числа привлеченных к разработке программистов число рабочих связей между ними N_c растет примерно пропорционально квадрату количества участников n или структурных единиц в ПО, исходя из правила один разработчик – одна структурная единица.



$N_c = n*(n-1)/2$

Рис.4. Нелинейный рост числа связей между участниками проекта

Реальная зависимость числа связей между частями разрабатываемого ПО, также как и трудоемкость их установления и отладки при разработке ПО несколько меньше квадратичной. Действительно, при разработке ПО надо стремиться к минимальной связности разрабатываемых модулей –структурных единиц и реальная связность модулей и разработчиков далека от полной. Поэтому реальная зависимость этой части трудоемкости от числа участников

разработки – размера ПО слабее квадратичной и общая трудоемкость разработки ПО в зависимости от его размера может быть оценена степенной функцией вида [90].

$ТР=К*ОБЪЁМ^{E}$,
Здесь ТР– трудоемкость разработки ПО,
ОБЪЁМ – объём оцениваемого ПО
E=1,0 -1,5

Поэтому характеристика **«среднее число строк в день на человека» далеко не константа и зависит от размера ПО и его сложности, от особенностей среды разработки.** Представление её в виде всеобщей константы неверно. Либо эта константа должна быть представлена, как число с большим разбросом (см рис.3).

Средняя производительность программиста в литературе описывается определенной по статистике разработанных программ величиной 7-16 строк исходного текста на ЯП в день (такой большой разброс производительности вызван различной сложностью ПО и нелинейной зависимостью «трудоемкость – объем ПО»).

На первый взгляд здесь парадокс. Студент во время лабораторной работы за час сделает больше. Дело здесь не в большой работоспособности и умственных возможностях молодости, а в том, что студент занимается программированием на заданную тему. Написать кода в единицу времени при этом конечно можно на порядок больше. Но для того, чтобы начать писать реально полезную программу необходимо сначала изучить задачу, затем выбрать алгоритм решения, дальше нужно написанные строки отладить, снабдить документацией, сдать заказчику.

Опубликованная производительность у профессионалов - средняя по циклу разработки с учетом затрат, не связанных непосредственно с программированием.

Метод функциональных точек.

Создатели метода ФТ проделали большую работу по определению возможностей перевода размера ПО, оцениваемого количеством ФТ в размер ПО, оцениваемый количеством строк исходного текста. Опубликованы данные, определенные по статистике огром-

ного количества ПО на разных языках программирования. Эти данные характеризуют эффективность ПО, а так же дают ориентировку в среднем размере функциональной точки на разных ЯП.

Таблица1. Размер ФТ в числе строк исходного текста на ЯП

Язык программирования	максимальное количество строк исходного текста ЯП на одну ФТ
Ассемблер	330
C	128-150
Basic	105-107
Fortrun	105-107
Algol	105-107
Modula 2	80
Fortrun 95	71
Fort	64
Prolog	64
LISP	64
C++	64
Java	51
ADA95	49
Delphi	30
Visual Basic	32
Excel	6

Следует иметь ввиду различную избыточность машинного кода, для различных языков программирования и трансляторов с них. Эта избыточность связана с несовершенством компиляторов. В нашей таблице мы эту избыточность не учитывали. В таблице приведены строки исходного текста до компиляции.

Для того, чтобы определить размер программы в числе строк исходного текста достаточно определить число ФТ и умножить его на средний размер одной ФТ в выбранном языке программирования в соответствии с таблицей 1.

Составление концептуальной модели ПО для оценки размеров ПО методом ФТ.

Рис. 5 Схема проведения оценки размеров ПО по данным и транзакциям

Оценка программного продукта методом ФТ вырабатывается не на пустом месте – рассматриваются данные, которыми оперирует продукт, типы и количество его транзакций.- процессов преобразования данных (см. Рис. 5) [11].

К логическим данным оцениваемого программного продукта относятся:

- внутренние логические файлы **(ILFs)** – выделяемые пользователем логически связанные группы данных или блоки управляющей информации, которые порождаются и поддерживаются внутри оцениваемого программного продукта.

- внешние интерфейсные файлы **(EIFs)** – выделяемые пользователем или программным окружением логически связанные группы данных или блоки управляющей информации, которые использует продукт, но которые порождаются и поддерживаются вне продукта.

Количество файлов определяется раздельно для ввода и вывода информации из оцениваемого ПО, а также для пользователя и программного окружения.

Рассмотрим подробнее выполнение шагов метода ФТ[11].

1.Количество функциональных точек в программе оценивается по количеству логических файлов групп данных внешних и внутренних (RET) и уникальных полей данных в этих файлах (DET):

- входных данных от пользователя,

- входных данных от других приложений, работающих с рассматриваемым,

- выходных данных из продукта на пользователя,

- выходных данных из продукта на другие приложения,

-количеству внутренних логических файлов.

.По **количеству и структуре данных (**RETи DET) производится оценка их «сложности» в соответствии с приведенными в методе таблицами (смотри приложение). В результате данные оцениваются как низкой, средней или высокой сложности. **Оценки проводятся отдельно для внутренних файлов ILF и внешних файлов EIF.**

2. По проведенной оценки сложности данных определяется количество «не выровненных ФТ» (UFP) в соответствии со специальной таблицей., приведенной в методических материалах метода (смотри приложение).

3. Оценивается сложность транзакций. Транзакция переводит оцениваемый продукт из одного состояния в другое. Рассматриваются виды транзакций:

- изменяет поведение продукта (тип EI),

-поддерживает (управляет исполнением) внутренние файлы (тип EI),

- представляет информацию пользователю (тип EO)).

В соответствии с этим рассматриваются виды транзакций

EI (external inputs) — внешние входные транзакции, элементарная операция управления, имеющиеся в интерфейсе ПО и приходящие извне, по обработке данных,

EO (external outputs) — внешние выходные транзакции, элементарная операция по генерации данных или управляющей информации, которые выходят за пределы программы, представляя информацию пользователю или программному окружению. Предполагает определенную логику обработки или вычислений информации из одного или более ILF.

EQ (external inquiries) — внешние запросы, элементарная операция, которая в ответ на внешний запрос извлекает данные или управляющую информацию из ILF или EIF.

Под внешними запросами при расчете ФТ-оценок следует понимать диалоговый ввод, который немедленно приводит к немедленному программному ответу в виде диалогового вывода. Основным отличием запроса от пары ввод-вывод является отсутствие вычислений либо каких-то других сложных действия в рамках этой процедуры взаимодействия. Например, ввод данных о клиенте через диалоговую форму с последующим сохранением данных в БД является

вводом, вывод на экран отчета по активности клиентов за последние три месяца является выводом, а поиск клиента по наименованию является запросом.

Сложность транзакций оценивается как низкая, средняя, или высокая в соответствии с таблицей по видам транзакций(смотри приложение).. Оценка сложности транзакции основывается на следующих ее характеристиках:

FTR (file type referenced) — позволяет подсчитать количество различных файлов (информационных объектов) типа ILF и/или EIF модифицируемых или считываемых в транзакции.

DET (data element type) — число неповторяемых уникальных полей данных. Примеры. EI: поле ввода, кнопка. EO: поля данных отчета, сообщений об ошибке. EQ: поля ввода для поиска, поля вывода результата поиска.

4. По проведенной оценке сложности транзакций оценивается количество «не выровненных ФТ» в соответствии с таблицей, приведённой в материалах метода, по типам транзакций (смотри приложение).

5. Проводится подсчет суммарного количества «не выровненных ФТ» UFP суммированием по всем информационным объектам данных и транзакций.

6. Помимо функциональных требований концептуальной модели ПО на оцениваемый программный продукт накладываются общесистемные требования, которые увеличивают сложность разработки. Для учета этой сложности применяется коэффициент выравнивания (VAF). Значение VAF зависит от 14 параметров D_i, которые определяют системные характеристики оцениваемого программного продукта и приведены в специальной таблице метода(смотри приложение). и принимают значения каждый в диапазоне 0-5:

$$VAF = 0.01\sum Di + 0.65$$

Можно определить диапазон влияния факторов сложности на размер программного продукта, как диапазон изменения VAF, равный 0,65-1,35, т.е. более чем в 2раза. Таким образом, механизм выравнивания позволяет разработчику ПО учесть его возможные индивидуальные особенности.

7. Расчет окончательного количества «выровненных функциональных точек» (FP) продукта. Оценка количества выровненных функциональных точек для программы определяется путем умножения суммарного количества «невыровненных ФТ» UFP по п5 на «коэффициент выравнивания» VAF по следующей формуле:

FP = UFP × VAF.

8.Расчёт объёма программы в числе строк исходного кода на ЯП, проводится путём умножения определённого числа выровненных ФТ на размер одной ФТ на языке программирования из таблицы1.

Глава 3.Оценка трудоемкости программного проекта по его размеру Модели процесса разработки ПО и выбор адекватной модели.

Методика COCOMO11

Метод анализа функциональных точек оценивает размер программы, но ничего не говорит о трудоемкости разработки оцениваемого продукта. Вопрос решается просто, если компания разработчик имеет собственную статистику трудозатрат на реализацию функциональных точек. Если такой статистики нет, то для оценки трудоемкости нужно использовать метод COCOMO (COnstructive COst MOdel – конструктивную модель стоимости), позволяющий по размеру программы определить её трудоемкость, а также сроки исполнения программного проекта [5,8,11]. В 1997 методика была усовершенствована и получила название COCOMO II.. В модели используется формула нелинейной регрессии с параметрами, определяемыми на основе обработки статистики по большому количеству программных проектов методом наименьших квадратов. Формула оценки трудоемкости проекта **PM** в чел.*мес. имеет вид:

$$PM = A \times SIZE^E \times \prod_{i=1}^{n} EM_i$$

$$A = 2{,}94$$

$$E = B + 0{,}01 \times \sum_{j=1}^{5} SF_j$$

$$B = 0{,}91$$

где

$SIZE$ - размер продукта в KSLOC

EM_i - множители трудоемкости

SF_j - факторы масштаба

$n = 7$ - для предварительной оценки

$n = 17$ - для детальной оценки

Различаются две стадии оценки проекта ПО: предварительная оценка на начальной фазе и детальная оценка после проработки архитектуры. Разница между этими оценками связана с числом факторов среды разработки и трудоемкости, учитываемых в формуле.

Мы рассматриваем предварительную оценку, так как информации для детальной оценки на стадии планирования разработки может не хватать. Для того, чтобы оценить трудоемкость, необходимо знать размер программного продукта в тысячах строках исходного кода (KSLOC - Kilo Source Lines Of Code). Размер программного продукта может быть, например, оценен с применением метода функциональных точек смотри рис.5.

Факторы масштаба проекта и среды разработки

В методике используются пять факторов масштаба проекта SF_i, которые определяются следующими характеристиками проекта: наличием опыта подобных разработок, детерминированностью(гибкостью) процесса разработки, зрелостью процессов разработки по шкале CMM, наличием анализа рисков, сработанностью команды разработчиков. Каждый фактор масштаба оценивается по пятибальной шкале – очень низкий, низкий, средний, высокий, очень высокий уровни.

Значение фактора масштаба, в зависимости от оценки его уровня, приведены в специальной таблице. Если все факторы мас-

штаба оцениваются как очень низкие т.е. 0, E=B=0,91. Если все факторы масштаба будут оценены, как наивысшие т.е. их сумма равна 31,6 и E=1,33.

Для размера ПО 10 килослов исходного текста разница трудоемкостей для этих случаев будет отличаться примерно в 3 раза. Таким образом метод COCOMO довольно сильно изменяет оценки трудоёмкости в зависимости от факторов масштаба – индивидуальных характеристик программного проекта.

Для оценки влияния факторов среды разработки необходимо **оценить для проекта уровень семи множителей** трудоемкости Mi, характеризующие факторы среды разработки. Они сведены в специальную таблицу, приведённую в приложении. Среди факторов среды разработки необходимо отметить:

– квалификация персонала (оценивается 5 уровнями от Extra Low – аналитики и программисты имеют низшую квалификацию, текучесть больше 45%; до Extra High - аналитики и программисты имеют высшую квалификацию, текучесть меньше 4%),

– сложность и надежность продукта (оценивается 5 уровнями от Extra Low – продукт простой, специальных требований по надежности нет, БД маленькая, документация не требуется; до Extra High - продукт очень сложный, требования по надежности жесткие) ,

– разработка для повторного использования (также оценивается пятью уровнями),

– сложность платформы разработки (также оценивается пятью уровнями)

– опыт персонала (также оценивается пятью уровнями) и .т.п.

Влияние множителей трудоемкости в зависимости от их уровня определяется их числовыми значениями, которые представлены в матрице, приведенной в методике (смотри приложение).

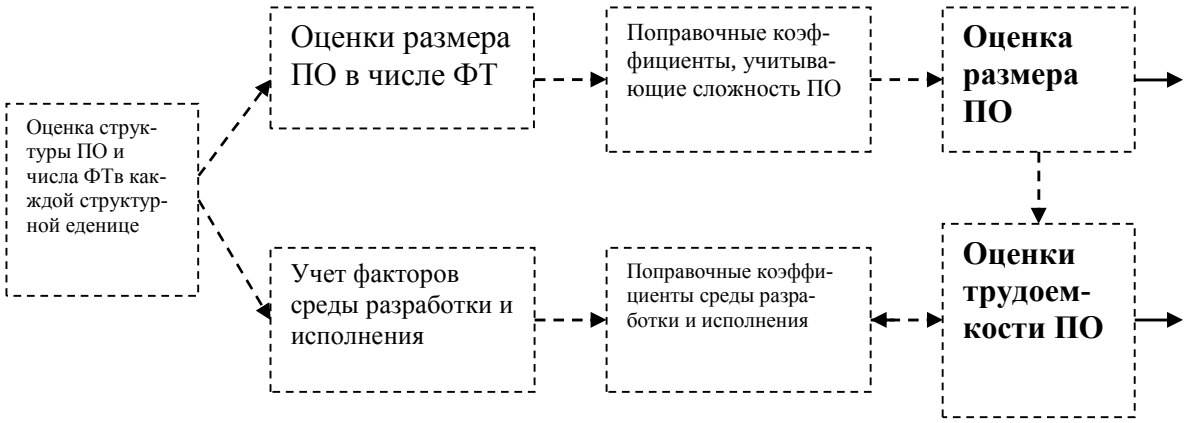


Рис.6 Схема определения трудоемкости разработки ПО, до начала его программирования

Оценка возможности реализации ПО в зависимости от размера в числе ФТ.

Имеется статистика по трудоемкости разработки программ, размер которых измеряется в функциональных точках. Статистика приведена для ЯП С со средним размером ФТ– 128. Оценки проведены для средних факторов среды разработки.

Для реализации проектов 1 ФТ требуется 1 день – это небольшая утилита. Эти работы всегда кончаются успешно. (1фт/чдн).

Размер ПО в 10 ФТ- это типичный объем небольших приложений и дополнений, вносимых в готовые программные системы. Такие работы так же всегда заканчиваются успешно, выполняются одним человеком и занимают до 30 дней (0.33фт/чдн.). Нелинейная зависимость трудоемкости от размера программного кода на лицо.

Размер ПО в 100 ФТ – это фактически предел программиста-одиночки. Этот процесс доводится до конца за 6 мес. и то только в 65% случаев.

Размер ПО в 1000 ФТ характерен сегодня для большинства коммерческих приложений. Заметную роль начинает приобретать документация. Для разработки необходима группа в 10 человек и около года времени(0.28фт/чдн.)Успехом работа заканчивается в 75% случаев. Если за такую работу берется программист-одиночка, то он справляется с нпей только в 35% случаев.

Размер ПО в 100 000 ФТ – это в настоящий момент предел возможностей технологии. Это размер ОС и крупных военных программных систем. Над их созданием работает более 200 чел. Работают от5 до 8 лет. До 65% проектов такого класса заканчиваются

неудачей из-за невозможности эффективно координировать работу такого количества человек (0,2 фт/чдн.).

Процессы разработки ПО и выбор адекватной модели

На долю программистов - разработчиков программ приходится не только относительно простая и рутинная работа по кодированию заданных математических моделей и задач на соответствующий язык программирования, но и куда более сложная и творческая работа по постановке и формализации задач, решаемых на компьютере с созданием соответствующих математических моделей, по созданию программы не только решающей функциональную задачу, но и способной к развитию и изменениям, удобной для отладки и устойчивой к ошибкам как собственных, так и внешней среды.

Существует большая разница между программированием заданных примеров с учебными целями и решением задач с использованием ПО, когда алгоритмов решения задач никто не задает и разработчику ПО самому требуется их найти и спроектировать, изучив предметную область, затем запрограммировать и убедиться, что выбранный алгоритм действительно решает поставленную задачу.

Все становится ещё более сложным, когда речь идёт о разработке большого ПО. Разработка больших систем ПО предусматривает нечто большее, чем просто определение и программирование алгоритмов выполнения составляющих ПО задач. Требуется разработать:

1.структуру такого ПО,

2.схему взаимодействия компонентов ПО между собой в «пространстве и во времени» и добиться работоспособности этой схемы.

3.меры по восстановлению работоспособности ПО и системы в случае возникновения нештатных ситуаций.

Нештатная ситуация – сочетание условий и обстоятельств при эксплуатации сложных технических систем и их ПО, отличающихся от предусмотренных для нормального функционирования и ведущих к возникновению аварийных и опасных состояний. В число нештатных ситуаций входят отказы оборудования системы, проявлению ошибок в программном обеспечении, ошибки во входных данных, отклонения от заданных условий эксплуатации, на которые проектировалась, отрабатывалась и испытывалась система.

Внешние возмущающие факторы (ВВФ), человеческий фактор, большое количество элементов и программ, каждые из которых имеет конечную надежность и с течением времени отказывают, являются причиной того, что ситуация, когда сложная техническая система (СТС) функционирует полностью исправной, является крайне редкой.

Поэтому управление работой ПО в нештатных ситуациях является обязательной частью разработки ПО для критических систем и существенно увеличивает трудоёмкость разработки. Оно устанавливает методы и дополнительные средства ПО для контроля его работы и работы системы с целью выявления проявившихся ошибок и отказов и восстановления работоспособности системы.

Существует также большая разница между программированием «для себя» или разработкой ПО в небольших научно- исследовательских лабораториях или частных предприятиях и разработкой ПО «на заказ» в промышленности в больших коллективах для критических систем, ошибка ПО в которых может привести к большим экономическим потерям или гибели людей.

В таких коллективах приходится управлять взаимодействием участников разработки, разделять ответственность за произведенный программный продукт, и здесь всегда есть люди, которые могут уволиться в любой самый неподходящий момент, а так же есть люди просто не желающие или не могущие работать.

В результате трудности, возникающие при разработке большого ПО растут далеко не пропорционально увеличению его функциональности и объему кода. Ясно, что в этом случае методы и средства разработки программ будут другими, чем в случае любительского программирования. Например, здесь без документирования программной продукции и процессов ее производства с фиксацией ответственности участников разработки не обойтись.

Глава 4. Планирование разработки ПО и системный подход к разработке ПО. Каскадная и спиральная модель жизненного цикла ПО. SW-CMM (Capability Maturity Model for Software)

Планирование и системный подход к разработке ПО. Каскадная модель жизненного цикла ПО

Для того, чтобы планировать разработку ПО необходимо создать **полный перечень** необходимых работ, выстроить их в порядке логического следования, после чего каждой позиции этой упорядоченной последовательности **назначить исполнителей** работ и **срок выполнения**, исходя из трудоемкости работы и имеющихся трудовых ресурсов.

При рассмотрении процесса разработки и определения полного перечня работ ПО мы будем придерживаться системного подхода, который предполагает рассмотрение не каких-то отдельных аспектов проблем конструирования и разработки ПО, а проблемы разработки ПО в целом. Системный подход лежит в основе понятия жизненный цикл разработки ПО, которое и определяет без деталей основной перечень и последовательность работ по ПО. Системный подход реализуется в «пространстве» и во времени.

Системный подход во времени рассматривается от момента формирования неудовлетворенной потребности в ПО до момента её разрешения, как последовательность обязательных этапов разработки.

Системный подход в "пространстве" предусматривает рассмотрение разрабатываемого ПО, программы, модуля, как части системы, работу которой оно обеспечивает. При этом на базе изучения информационных потребностей системы, в которую будет входить разрабатываемое ПО, формулируются цели и набор функций ПО, анализируются прототипы программных средств. Формируются и документируются требования к ПО.

Современная технология разработки ПО рассматривает программирование - создание программного кода для ЭВМ, как один из этапов разработки в цепи последовательных этапов цикла разработки. Все эти этапы объединяются понятием жизненный цикл ПО и должны быть поддержаны соответствующими инструментальными программными и аппаратными средствами.

Понятие жизненный цикл является отражением системного подхода к разработке ПО. В соответствии с международным стандартом ISO /IEC 12207 «информационные технологии – Процессы жизненного цикла ПО» имеются следующие этапы жизненного цикла ПО

1) анализ системных требований и области применения;

2) проектирование архитектуры системы;

3) анализ требований к ПО (спецификации, внешние интерфейсы, требования к квалифика ционным испытаниям);

4) проектирование архитектуры ПО с выделением структурных частей ПО и связей между ними;

5) детальное проектирование каждой структурной единицы ПО;

6) кодирование ПО (программирование)

7) тестирование и отладка структурных единиц ПО;

8) интеграция (объединение ПО) и тестирование совокупности структурных единиц ПО;

9) квалификационные испытания ПО (комплексная отладка);

10) интеграция системы единицы структуры ПО должны быть объединены с единицами аппаратных средств;

11) квалификационные испытания системы;

12) установка ПО.

Таким образом, процесс разработки ПО имеет свое начало от системы, где это ПО будет использовано (этапы1,2) и завершается опять в системе (этапы10,11,12). Этапы 3,4,5 относятся к этапам проектирования ПО. Этапы 7,8,9 – к этапам отладки ПО.

После этапов разработки в жизненном цикле ПО следует этап эксплуатации ПО и его сопровождения при эксплуатации.

Данная модель жизненного цикла (ЖЦ) относится к модели каскадного типа. Этот тип модели ЖЦ хорош для ПО, для которого в самом начале разработки системные требования хорошо проработаны и возможно полно и точно сформулировать все требования к ПО. Однако, реальный процесс создания ПО не всегда укладывается в такую жесткую последовательную схему и часто возникает необходимость возврата к уже пройденным этапам с уточнением или пересмотром уже принятых прежде всего системных решений из-за изменений требований заказчика, изменений в аппаратуре, обнаружения принципиальных ошибок в построении ПО.

Каскадную модель, конечно, можно к этому приспособить, но лучше с самого начала для ПО такого типа рассматривать модель

жизненного цикла, «заточенную» под быстрые итерации для получения недостающих требований к ПО.

Спиральная модель ЖЦ ПО.

Для некоторого типа программных систем первоначальные требования, к которым недостаточно определены, итеративный процесс разработки ставится во главу угла.

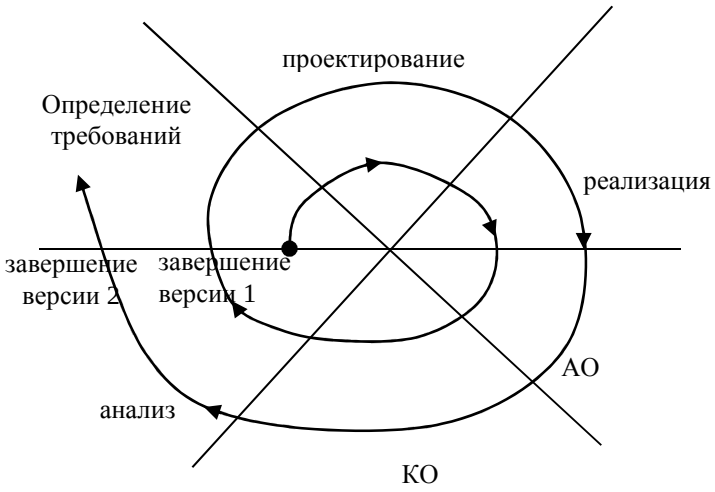


Рис.7 Схема спиральной модели ЖЦ ПО

Главная задача как можно быстрее достичь работоспособного ПО, активизируя тем самым процесс уточнения и дополнения требований. Это так называемая спиральная модель ЖЦ ПО, изображенная на рис.7.

Эта схема ЖЦ для ПО, некачественные первые версии которого допустимы по функциональному назначению ПО.

Модели процессов разработки ПО принято классифицировать на «тяжелые и легкие» по количеству формализованных процессов (большинство процессов или только основные) и детальности их регламентации. Чем больше процессов регламентировано и документировано, чем более детально они описаны, тем больше «вес» модели.

В стандарты разработки ПО заведены тяжелые каскадные модели ЖЦ и на основе этих стандартов разрабатываются программные системы по госзаказам в России. Стандартов на легкие технологии разработки ПО, которые разрабатываются в основном по спиральной модели, отсутствуют.

Управление изменениями программного проекта

Управление изменениями является неотъемлемой частью управления программными проектами. Неизбежные в процессе производства и эксплуатации изменения в документации на ПО, связанные с устранением ошибок, развитием и улучшением ПО вносятся в документацию на ПО (код) только по специальному и формальному документу, который называется извещением на изменение. Этот процесс несколько по - разному происходит для ПО, которое сдано заказчику и находится в эксплуатации и для ПО, которое заказчику ещё не сдано.

С целью предотвращения несанкционированных и не скоординированных изменений одна из копий ПО объявляется подлинником и отчуждается от разработчика путём помещения в специальный архив подлинников. В этом случае вводится «служба архива подлинников», в которой документы ПО, объявленные подлинниками, открыто доступны только для чтения. Изменения в документы ПО, находящиеся в архиве подлинников ПО, возможна только по санкции руководителя проекта ПО. Эта санкция оформляется документально, например, в виде «Решения на доработку ПО» и в ней указывается причины, характер, объем и сроки проведения коррекции ПО, необходимость коррекции смежных программ ПО и документации на систему, объем и сроки необходимой отладки. По завершению проведения изменений в соответствии с «Решением на доработку» измененное ПО внедряется в систему путём выпуска упомянутого извещения на изменение. Без выпуска извещения коррекция ПО недопустима.

Операция «отчуждение подлинника» проводится после завершения автономной отладки фрагментов ПО и передачи ПО на комплексную отладку. После сдачи ПО в эксплуатацию уровень санкции на изменение ПО должен быть повышен и может быть максимально высоким [6].

В изменяемом документе - подлиннике ПО фиксируются все изменения, кто и когда их произвел.

Такое отчуждение разработчика от своей документации позволяет сделать все изменения в документации наблюдаемыми и санкционированными.

При этом технология подготовки «решения на доработку ПО» и извещения на изменения какой-либо части ПО должна быть такова,

что для их утверждения требуется согласование со всеми заинтересованными сторонами, на которые эти изменения могут повлиять. Кроме того, в извещении указывается, что делать с заделом уже произведенного и поставленного ПО, нужно ли и его дорабатывать. Этим обеспечивается защита от изменений, способных нарушить связную работу изменяемого ПО в целом.

После проведения изменений в подлиннике они тиражируются и рассылаются этим же техническим архивом во все учтенные копии. Отсутствие рассылки изменений документации в одно или несколько мест нахождения учтенных копий приводит к тому, что документация в этих местах (эксплуатации или производства) становится неправильной - расходится с правильной документацией подлинника.

Принцип "отчуждения подлинника" от разработчика является развитием широко применяемого в промышленности принципа ведения разработанной технической документации на производство и эксплуатацию сложного изделия или системы.

Тяжелые и легкие технологии разработки ПО.

Тяжелые и легкие модели производственного процесса имеют свои достоинства и свои недостатки. В основе тяжелых процессов разработки лежит каскадная модель жизненного цикла. **Тяжелые технологии** разработки характеризуются жесткой регламентацией т.е. порядком разработки. Они требуют глубокого документирования процессов и существенной управленческой надстройки. Они обладают более длительными стадиями анализа и проектирования, более формализованными рабочими коммуникациями между участниками проекта. В тяжелых технологиях работает установленный регламент–порядок разработки, который надо всё время поддерживать.

Тяжелые технологии рассчитаны на среднюю квалификацию исполнителей и большую их специализацию. В этих случаях ниже требования к стабильности команды разработчиков. Отсутствуют ограничения по масштабу проекта.

Легкие (подвижные) технологии разработки(**Agile).** Они основываются на спиральной модели жизненного цикла ПО. Основная идея облегченных моделей заключается в том, что применяемый в разработке ПО технологический процесс должен быть не жестко

регламентирован, а адаптивным. Эти технологии декларируют своей высшей ценностью ориентированность на людей и их взаимодействие, а не на регламент процесса и средства. Главный принцип: не люди должны строиться под выбранную модель процесса разработки, а модель процесса должна подстраиваться под конкретную **(располагаемую)** команду, чтобы обеспечить её наивысшую эффективность. В легких процессах проповедуются неформальные коммуникации между участниками работ. Легкие процессы разработки обеспечивают упрощенные стадии анализа и проектирования, основной упор делается на разработку. Поэтому эффективность процессов разработки сильно зависит от индивидуальных способностей, требуют более квалифицированной, универсальной и стабильной команды. Объем и сложность выполняемых проектов ограничены.

Не существует единственного правильной технологии разработки ПО, в каждом новом проекте процесс должен определяться каждый раз заново, в зависимости от трех П: **проекта, продукта и персонала**.

Модель зрелости разработки ПО (SWCMM - Capability Maturity Model for Software)

В середине 80-х годов минувшего столетия Министерство обороны США, столкнувшись частыми провалами в реализации программных проектов, крепко задумалось о том, как выбирать разработчиков ПО, какими свойствами характеризуются правильные разработчики ПО. По заказу военных Институт программной инженерии разработал SWCMM в качестве эталонной модели организации разработки программного обеспечения.

Данная модель определяет пять уровней зрелости процесса разработки ПО [9].

1. Начальный — процесс разработки носит хаотический характер. Определены лишь немногие из процессов, и успех проектов зависит от конкретных исполнителей.

2. Повторяемый — установлены основные процессы разработки и управления проектами: отслеживание затрат, сроков и функциональности. Упорядочены–документированы некоторые процессы, необходимые для того, чтобы повторить предыдущие достижения на аналогичных проектах.

3. Определенный — процессы разработки ПО и управления проектами описаны и внедрены в единую систему процессов компании. Во всех проектах используется стандартный для организации процесс разработки и поддержки программного обеспечения, адаптированный под конкретный проект.

4. Управляемый — собираются детальные **количественные данные по функционированию процессов разработки и качеству конечного продукта**. Анализируется значение и динамика этих данных. Здесь уже проглядывается научно-исследовательский подход к разработке ПО

5. Оптимизируемый - постоянное улучшение процессов основывается на **количественных данных** по процессам и на пробном внедрении новых идей и технологий.

Документация с полным описанием SWCMM занимает около 500 страниц и определяет набор из 312 требований, которым должна соответствовать организация, если она планирует аттестоваться по этому стандарту на 5-ый уровень зрелости.

У данной модели два назначения.

1.Помочь заказчику ПО выбрать организацию, которая может успешно справиться с проектом ПО.

2.Разработчикам ПО оценить эффективность своих рабочих процессов и определить области и направления улучшений.

Модель компетентного разработчика PSP

PSP (Personal Software Process) - **о**дна из последних разработок Института программной инженерии, которая определяет требования к компетенциям разработчика ПО. Согласно этой модели каждый программист должен уметь:

- учитывать время, затраченное на работу над проектом;
- учитывать найденные дефекты, классифицировать типы дефектов;
- оценивать размер задачи;
- планировать программные задачи, распределять их по времени и составлять график работы.
- выполнять индивидуальную проверку проекта и архитектуры;
- осуществлять индивидуальную проверку кода;
- осуществлять систематический подход к описанию вариантов и результатов

тестирования;

Последовательное применение модели PSP к разработчикам ПО позволяет сделать нормой в организации пятый уровень CMM.

Глава 5. Декомпозиция (разбиение) СТС и ПО на подсистемы – универсальный метод снижения сложности разработки. Аутсорсинг. Организация разработки в большом. Организационная структура компании разработчика ПО.

Декомпозиция и аутсорсинг

Одна из первых работ проектировании системы – декомпозиция её на подсистемы определенного функционального назначения. Декомпозиция снижает сложность проектирования, а затем сложность разработки и эксплуатации СТС. За декомпозицией стоит не только уменьшение сложности проектирования за счет абстрагирования от сложности целого при разработке каждой выделенной части, декомпозиция – разделение труда, поскольку за каждой выделенной частью должен стоять свой разработчик. Это позволяет проводить разработку частей – подсистем параллельно во времени, что сокращает сроки разработки.

Но подсистемы в процессе работы системы должны взаимодействовать и функционировать как единое целое. В настоящее время объединение подсистем происходит на уровне ПО системы, в котором каждой подсистеме соответствует своя часть. Именно по подсистемам происходит разработка ПО. Объединение в единое целое осуществляется на программном уровне. ПО СТС реализуется либо в единой центральной системной ЦВМ, либо распределенным образом в ЛВС, включающей в себя системную ЦВМ и периферийные ЦВМ, встроенные в подсистемы. Поэтому, разбивая сложный объект или ПО на части, надо всегда иметь в виду возможность последующей легкой сборки целого.

При этом следует отметить, что разделение СТС на части - это эвристическая процедура, которая определяется прежде всего проникновением разработчика в природу задачи, его квалификацией и изобретательностью.

Части целого - подсистемы разрабатываются специалистами разного профиля параллельно, разработка подсистем больших систем часто проводится с использованием аутсорсинга – передачей на

сторону для разработки в специализированные организации отдельных подсистем системы. Этот процесс требует со стороны головной организации выдачи ТЗ соисполнителям и приемки готовой подсистемы про специальной программе приемо - сдаточных испытаний. Другая основная роль головной организации – сборка и испытание целого – системы из разработанных порозень подсистем.

При этом при разработке подсистем возникают внутренние проблемы, не только прерывающие процесс разработки подсистем, но и требующие внесение изменений в весь проект и в другие подсистемы. Все это и определяет итеративный характер проектирования системы и ПО. Это ведет к изменениям требований к ПО и к изменениям в проекте ПО. Это должно делаться головной организацией путем уточнений соответствующих ТЗ.

Во всех случаях **декомпозиция приводит к возможности параллельной во времени разработке системы по подсистемам в чем и состоит одна из главных целей декомпозиции и аутсорсинга.**

Аутсорсинг широко применяется в промышленности и позволяет получить необходимое качество продукции при сокращении расходов за счет привлечения специализированно организации со специалистами с высокой квалификацией и опытом работы в той или иной сфере,

В тоже время аутсорсинг имеет и недостатки: риск утраты персоналом головной организации уникальных знаний и опыта в разработке подсистем, необходимых для разработки ТЗ и приемки работ, и сложность вследствие этого осуществления контроля работ соисполнителей подсистем,

Организация разработки ПО. Организация разработки в большом

В схеме разработки ПО нами наряду с действующими лицами: разработчиком ЦВМ, разработчиком ОС, разработчиком системы программирования, разработчиками прикладного ПО, пользователями необходимо отметить ещё одного важнейшего участника – системщика или постановщика задач для ПО. Здесь, конечно, везде имеются в виду не столько физические лица сколько соответствующие подразделения в данном случае системщиков или постановщи-

ков задач ПО. Эти люди владеют предметной областью, алгоритмом(алгоритмами) работы системы и наиболее тесно взаимодействуют с пользователем, часто им и являясь.

В настоящее время нередко разработчик ПО и системщик – это одно и то же действующее лицо, но бывает и так, что они организационно разнесены. Причем первое является желательной схемой разработки.

Поэтому организация разработки «в большом» соответствует тому или иному варианту разрешения основного вопроса системного подхода разработки ПО – кто разрабатывает ПО: разработчик системы или разработчик только ПО, нанятый разработчиком системы по аутсорсингу? В зависимости от ответа на этот вопрос решаются практически все этапы жизненного цикла ПО, начиная с определения требований и кончая сопровождением в эксплуатации.

Но есть ещё вопрос организации разработки, связанный с разделением труда уже внутри разработчика ПО. Этот вопрос возникает тогда, когда одновременно организация разработчика ПО, кем бы он не был, разрабатывает несколько проектов ПО. Организация разработки ПО «в малом» рассматривает формы разделения труда между проектами и функциями исполнителей. Эти формы: проектная, функциональная и матричная могут соответствовать обоим вариантам организации разработки в большом.

Таким образом, несмотря на возможные вариации в организации работ при разработки больших ПК СТС, используется одна из двух основных организационных схем:

- **Первая схема** – разработчик системы разрабатывает ПО для неё.

- **Вторая схема** – разработчик системы отдает разработку ПО в руки специального подразделения разработчика ПО (внутри организации) или организации разработчиков ПО, используя аутсорсинг.

Системщику, знающему систему, в которой должно работать ПО, и одновременно являющемуся разработчиком ПО, **легче определить, что же в системе возложить на ПО, а что будет делать «железо».** В этом смысле организация разработки по схеме 1 предпочтительнее, так как в ней более эффективно решаются основополагающие фазы жизненного цикла ПО (анализ системных требований, выбор требований к ПО, выбор структуры ПО).

Другие преимущества организационной схемы 1 перед схемой 2 связаны с отсутствием стыка «алгоритм-программа», который «висит» на полноте документа на программирование - ТЗ на программу. Этот документ в общем случае всегда страдает неполнотой, так как составить полный перечень того, что должна делать программа - это фактически ее написать. Чего как раз разработчик системы делать не хочет или не может. Из-за неполноты ТЗ на программу у разработчика ПО, не владеющего знаниями в предметной области, пониманием, что должна делать программа, может получиться программный продукт **неработоспособный, но соответствующий не точному и не полному ТЗ.**

Недостаток схемы 1 связан с возможной низкой программисткой квалификацией разработчиков систем. Однако, современная технология разработки уменьшает этот недостаток, так как в помощь разработчику предоставлены удобные инструменты - технологические среды разработки ПО с универсальными либо проблемно-ориентированными языками программирования.

Таким образом, развитие технологии разработки ПО стимулирует развитие удобных инструментов создания программ с графическими языками не требующие высокого профессионализма в программировании от системщика-разработчика ПО.

В обоих организационных схемах 1 и 2 должно присутствовать головное подразделение по ПО - бригада главного программиста. Это – подразделение комплексных работ по ПО в целом, которое определяет:

-состав и структуру ПО;

-единый план-график разработки ПО СТС по всем программам, согласованный по срокам с разработкой СТС.

-технологию разработки ПК, необходимые инструментальные средства;

-проводит комплексную отладку ПО;

-отчетные документы (отчеты) по выполнению основных этапов технологии определенной формы, делающие визуально видимым и контролируемым объем проведенных по этапу работ;

-документы, описывающие распределение работ и границы ответственности подразделений в процессе создания ПО;

Факторы успеха проекта. Норма управляемости. Управление проектами.

Нельзя рассматривать эффективность работы организации над проектом, исходя только из результатов работы персонала по принципу: чем больше работы сделано, тем выше эффективность. С таким подходом к проектной деятельности можно «зарезать на ужин курицу, несущую золотые яйца». Создание и закрепление эффективной команды разработчиков ПО – это стратегическое приобретение IT компании или проектной организации. Обучение участников проекта – инвестиции в будущее. Уход из компании всех профессионалов после проекта, выполненного по принципу «любой ценой», – затраты очень тяжело восполняемые. Главный капитал проектной организации или современной IT компании – это знания и люди, как носители знаний и умений.

Поэтому согласно упомянутому руководству по управлению программными проектами PMBOK, у проекта разработки ПО сегодня не три, а четыре фактора успеха:

1. Выполнен в соответствие с требованиями ТЗ и спецификациями.
2. Выполнен в срок.
3. Выполнен в пределах бюджета.
4. **Каждый участник команды уходил с работы во время с чувством успеха.**

Этот четвертый фактор успеха должен стать воспроизводимым, если предприятие хочет быть эффективным и продолжать создавать успешные проекты (на 4-5 уровне CMM). Для успешного программного проекта должно быть характерным постоянное ощущение его участниками чувства удовлетворения и гордости за результаты своей работы, чувства оптимизма. Нет ничего более гибельного для проекта, чем **равнодушие или уныние** его участников.

Структура организации должна обеспечивать её устойчивое и управляемое функционирования на базе разделения труда, оформленного в виде специализации подразделений. Зачем нужны подразделения? Зачем нужны начальники? Каковы их функции? Сколько нужно начальнику иметь подчиненных?

При прямом управлении группой людей и увеличении группы свыше 7 человек у управляющего возникают трудности. С ростом

размеров группы его нагрузка может стать чрезмерной, с которой он не справится. В этом случае иерархическая структура уменьшает нагрузку на руководителя и улучшает управляемость, правда, приводит к росту числа руководителей (смотри рис.8).

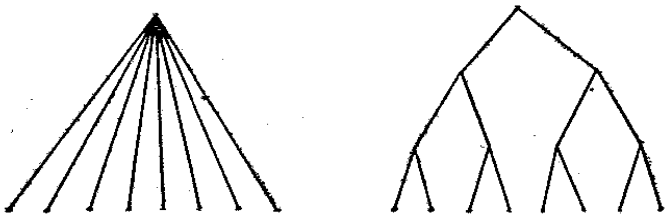


Рис.8 Иерархические структуры управления

Группа людей на одном уровне иерархии и образует подразделение, управляемое руководителем подразделения. В свою очередь группа руководителей первого ранга образует уровень иерархии, над которой стоит свой руководитель и т.п. Наличие подразделений в организации требует разграничения зон ответственности. Это должно закрепляться соответствующими организационными положениями, которые выпускаются в виде документа. Часто их отсутствие приводит к конфликтам. Этого допускать нельзя.

Завышенная норма управляемости для руководителя (числа его подчиненных) приводит к потере управляемости и дезорганизации управления. Число подчиненных на одного начальника, конечно, зависит от типа производства: в строительстве и другой операционной деятельности эта норма может быть 10 человек, а в проектной деятельности – 5-7 человек. Норма управляемости зависит от оснащенности подразделения средствами оргтехники. И, наконец, норма управляемости зависит от места подразделения в иерархической структуре, знаний и опыта руководителя. Великий полководец и организатор Александр Македонский делил своё войско на пятерки.

Приведенные соображения по норме управляемости приводят к иерархической структуре проектной организации с численностью сотрудников на каждом уровне иерархии, не превышающей 5-7 человек. Примерно столько же кафедр на факультете. Но иерархическое разделение по подразделениям можно провести различным образом. Например, в организации, разрабатывающей несколько про-

ектов, а успешно работающие организации растут численно и достигают больших размеров, можно собрать в одном подразделении, специалистов разных профилей, выполняющих работу по одному проекту, либо собрать в подразделении всех узких специалистов одного профиля, каждый из которых работает в рамках своего проекта. В первом случае это проектная форма организации, во втором – функциональная.

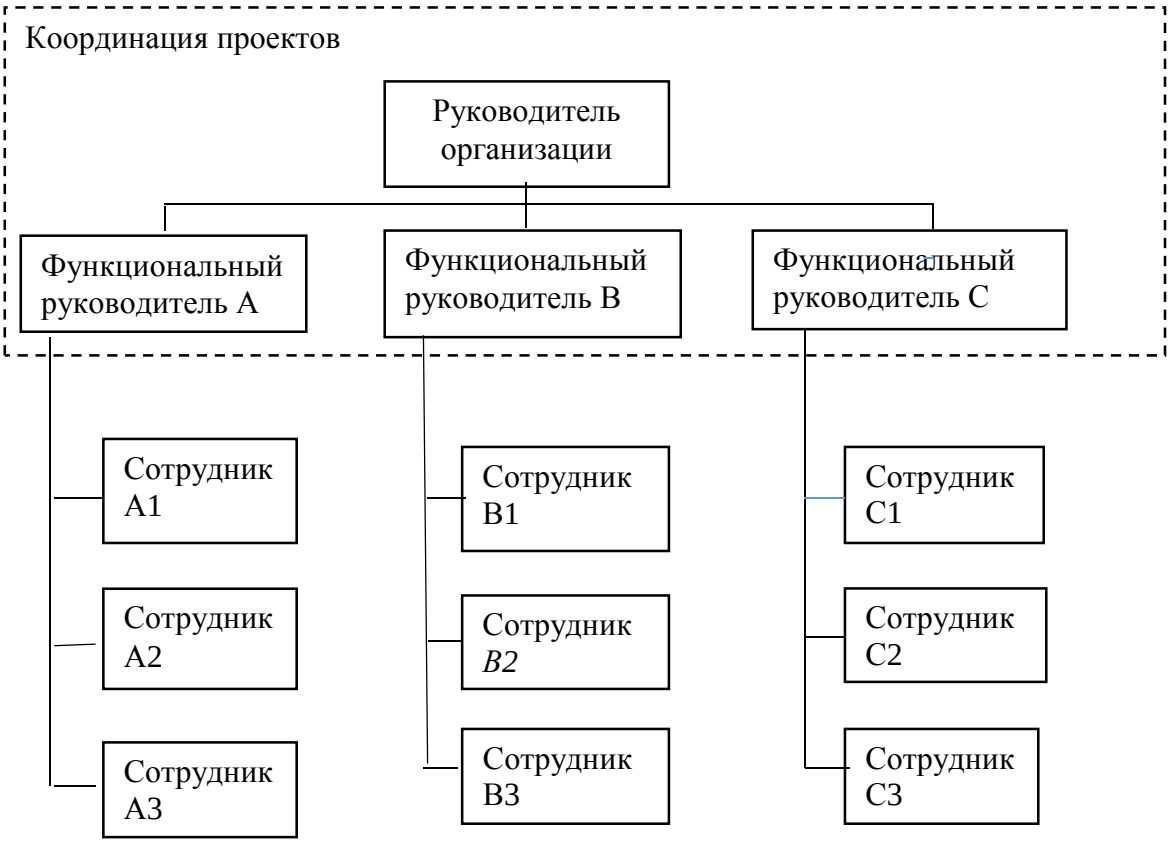


Рис.9 Функциональная структура организации.

Функциональная и проектная формы организации используются для управления проектами в том числе и программными – противоположные полюса, а матричная организация – широко применяемая промежуточная форма организации. Из этого следует, что нет одной лучшей организационной структуры на все случаи жизни.

Нет смысла противопоставлять функциональные структуры и проектные структуры для организации. Надо знать об их возможностях и применять ту, которая более подходит для конкретного проекта.

Если в организации разрабатываются, например, 4 проекта, то на каждый проект выделяются внутри подразделений по сотруднику и координация проекта производится руководителем организации. Например, проект1- А1,В1,С1; проект2 – А2,В2, С2; проект3 – А2,В3,С3; проект4 – А3,В1, С2.

Функциональная структура имеет следующие особенности:

1. Сохраняется принцип единоначалия
2. Понятные и стабильные условия работы
3. Управление проектом сконцентрировано и держится на компетенции высшего руководства. Затруднено принятие решений и коммуникации между исполнителями в рамках одного проекта. Осуществляются только через руководство высокого уровня, которое труднодоступно.

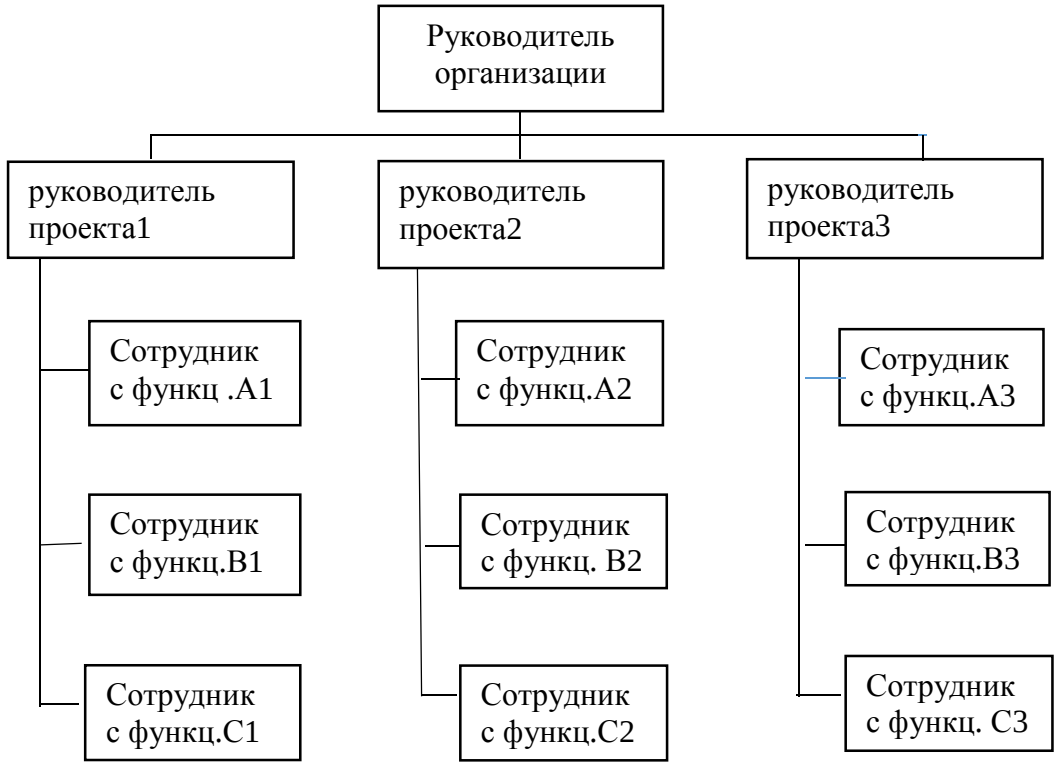


Рис.10 Проектная структура организации

4. Как правило, неэффективен контроль в функциональных подразделениях за ходом проекта (функциональный руководитель не имеет целостной картины проекта).

Функциональная структура предполагает многоуровневую иерархию. Руководители функциональных подразделений это начальники управлений, начальники подчиненных им служб, отделов, лабораторий, секторов, групп. А еще у каждого начальника есть заместитель и, порой, не один. Примеры: министерства, ведомства, научные институты и предприятия.

Проектная форма структуры организации

На другом краю спектра организационных структур находится проектная структура (Рис.10).

В чисто проектных организациях:

1. Проект организуется в самостоятельном производственном подразделении, в котором имеются работники разного функционального направления.
2. Персонал на проект набирается по временным контрактам.
3. После завершения проекта персонал увольняется, если нет другого проекта.
4. Команды не сохраняются. Опыт не аккумулируется.

Проектная структура организации не самая эффективная, но порой единственно возможная организационная форма для выполнения проектов. Преимущества её – мобильность и сведение к минимому недозагруженности и простоев. Если работа завершена, а других заказов нет, то команда расформировывается.

Но в последствии набор необходимой команды может составить проблему (проверенные и опытные люди ушли в другие проекты). Поэтому желательно заблаговременно определяться для хорошей стабильной **команды с новым проектом**. Не всегда мобильность проектных структур можно просто реализовать. Для этого нужна достаточно универсальная предметная область и универсальное оборудование и ПО. Главный недостаток проектной структуры – люди, работающие в ней, чувствуют себя неуверенно и не имеют мотивации для быстрейшего завершения проекта, так как за завершением проекта дальше – неопределенность. При такой организации неуверенно чувствуют себя и заказчики, так как не ясен процесс сопровождения в эксплуатации закупленного продукта.

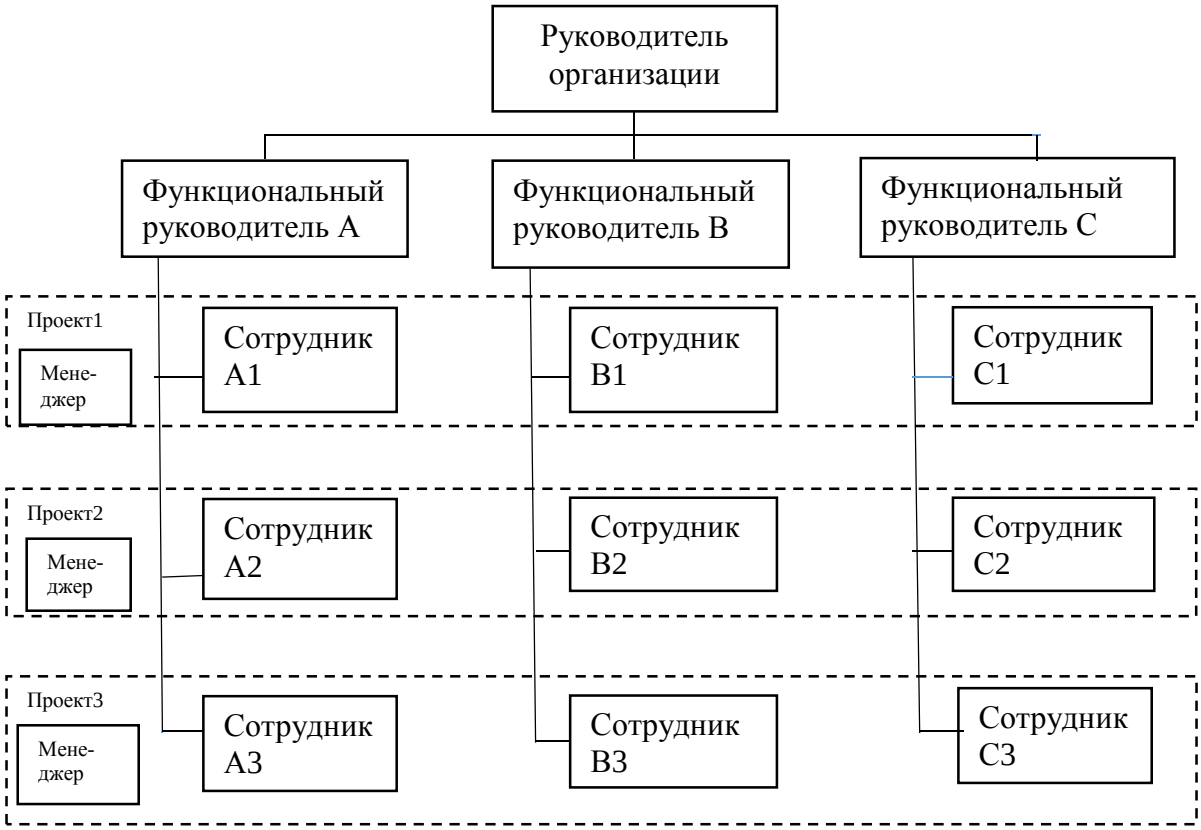


Рис.11 Матричная структура организации

Матричная форма структуры организации

Матричная организация характеризуется тем, что наряду с руководителями функциональных подразделений появляется менеджер того или иного проекта, который реально управляет выделенными на проект ресурсами (смотри рис.11). Он планирует работы, распределяет задачи среди исполнителей, контролирует сроки и результаты, несет полную ответственность за достижение целей проекта, при соблюдении ограничений.

В сбалансированных матрицах проявляется проблема «двойного подчинения». Руководитель функционального подразделения и менеджер проекта имеют примерно равное влияние на материальный и профессиональный рост разработчиков. Однако, формальное административное подчинение остается за функциональным начальником и принцип единоначалия не разрушается.

В разработке ПО наиболее распространены проектная и матричная организация. В IT компаниях, которые ориентированы на заказную разработку ПО, часто используется функциональная организа-

ция и функциональные подразделения чаще образуются в соответствие с используемыми информационными технологиями. Например, отдел разработки баз данных, отдел разработки приложений, отдел веб-разработок, отделы тестирования, документирования и т.д.

Глава 6. Планирование разработки ПО. Сроки разработки

Зачем надо планировать разработку ПО

В реальной жизни невыполнение намеченных планов – обыденная ситуация. Это касается не только ПО. Возникает вопрос: зачем планировать, если запланированные сроки часто всё равно срываются, запланированных ресурсов практически всё равно не хватает и предусмотренный бюджет будет трещать по швам? Стоит ли на планирование тратить время и средства?

Стоит потому, что:

1. Во - первых, нужно получить заказ, а для этого надо убедить заказчика в том, что с вами можно иметь дело. Как это сделать?

- Заявить: «Да мы здесь все кандидаты и доктора наук!» - малоубедительно.

- Заявить: «Мы уже делали что-то подобное» - звучит несколько лучше.

- Заявить: «У нас есть обоснованный план, следуя которому задача будет выполнена!» - это уже предмет для дальнейшего разговора, но этот обоснованный план должен быть достаточно подробным и его придётся предъявить и защищать.

2.Ход проекта должен быть контролируемым. План – инструмент визуализации контроля хода проекта.

3.Проект должен быть предсказуемым по затратам и времени выполнения. Как сделать его предсказуемым? План – один из основных элементов предсказуемости проекта. Контроль хода выполнения плана позволяет оценить возможность продолжения проекта. Оценить необходимый объем дальнейшего финансирования.

Проект всегда имеет элементы неопределенности. Т.е. заранее в сложном проекте всё предусмотреть или предсказать нельзя, в силу чего первоначальные планы обычно и не выполняются. Но при возникновении ранее неучтенных обстоятельств, приводящих к срыву плановых сроков, планы можно и нужно корректировать. Для сохранения предсказуемости проекта продолжать планировать работы

надо обязательно, так как **план – не догма, а инструмент управления.**

З*адачи планирования. Когда начинать планировать?*

Основными функциями планирования являются:

1.Преобразование потребностей в управляемые задачи.

Изначально проект выступает в виде требований, разработанных и согласованных с заказчиком. Цель планирования – представить его в виде совокупности отдельных задач, выполнение которых можно контролировать.

2.Определение необходимых ресурсов.

Детальные планы позволяют определить количество людей, необходимого оборудования и рабочие условия, которые понадобятся для выполнения проекта

3.Координация командной работы над проектом.

При планировании выполнение проекта разбивается на отдельные работы, которые можно выполнять параллельно. Это сокращает сроки. Планы делают возможной координацию путем определения того кто, что и когда делает.

4.Оценка потенциальных рисков.

Хотя некоторые риски могут быть выявлены во время формулировки требований, гораздо больше их обнаруживается после осуществления детального планирования. Знание о существовании этих рисков (нештатных ситуаций при выполнении запланированных процессов) позволяет раньше заметить их проявление и приготовиться к их адресации.

5.Сигнализация о возникновении проблем. Отклонение сроков выполнения работ от плана – сигнал о возникновении проблемы. Если выполнение проекта не оправдывает ожиданий, то необходимо провести соответствующую корректировку плана.

В относительно небольших проектах план может быть единым. В больших проектах наряду с единым планом могут составляться планы по отдельным видам работ (процессам): план тестирования, план документирования, план управления качеством, финансовый план и т.д.

На вопрос: «Когда необходимо начинать планировать работы проекта»? может[я1] быть несколько ответов:

-Когда сформулированы требования и ясен объем работ.

-Когда выполнение проекта выходит из под контроля и проект надо «ввести в берега».

Из этих ответов правильный первый. Если у разработчика ПО небольшая, слаженная команда профессионалов, то (см. технологию XP) планирование может быть сведено к разумному минимуму. Если разрабатывается большой проект, имеется большая команда, ограниченные сроки, то детальные планы придется составлять с самого начала.

Можно спросить: как планировать то, что пока еще не известно? Это не корректный вопрос, так как и в начале программного проекта о ПО многое уже известно - есть функциональное назначение, есть системные требования, есть данные на входе и данные на выходе, есть концептуальная модель ПО, есть опыт создания прототипов или подобных проектов, есть приближенные модели оценки характеристик проекта .

Здесь следует вспомнить, что планирование так же как и сама разработка –процесс циклический (итеративный). Это процесс составления, оценки и корректировки плана по получению первых результатов.

Планирование от трудоемкости разработки. Сетевые графики и их топология

Управление разработкой программных систем (Software managment) – это деятельность, направленная на организацию, планирование и контроль работы участников разработки ПО, на мотивацию их деятельности с целью выполнения треугольника ограничений, сопровождающих каждую инженерную разработку: **обеспечить требуемые функции и качество ПО, сроков и бюджет разработки ПО.**

При этом именно трудоемкость является тем комплексным параметром, от которого зависит и качество, и бюджет, и сроки. Большой разброс этой характеристики связан ещё и с тем, что трудозатраты на разработку ПО связаны с размером ПО и растут далеко не пропорционально размеру ПО.

При составлении плана разработки ПО за основу берутся этапы жизненного цикла ПО. Однако план должен быть подробным и включать в себя гораздо больше позиций, чем два десятка позиций

той или иной модели жизненного цикла. Например, в рассмотренных моделях жизненного цикла ПО отсутствуют позиции о необходимости разработки тестов для отладки. Отсутствует также позиция о подготовки драйверов для автономной отладки и моделей внешней среды для проведения комплексной отладки, которые необходимы для некоторых типов СТС. Отсутствуют позиции для подготовки эталонов всё той же отладке, поскольку только при сравнении результатов отладки с заранее подготовленным эталоном можно сделать суждение о наличии или отсутствии ошибки. Отсутствует ряд работ по проектированию ПО в части выпуска необходимых проектных документов весьма специфичных для ПО различного назначения.

Эти работы в универсальный укрупнённый перечень работ жизненного цикла не попали, так как они привязаны к конкретной технологии разработки ПО, к ПО определённого назначения, но в каждом конкретном случае необходимы.

В плане разработки конкретных видов ПО эти позиции в конкретном варианте используемой технологии должны быть приведены, также как позиции жизненного цикла должны быть проведены по каждой единице структуры ПО, по каждому предусмотренному к выпуску документу и во всяком случае по каждому структурному подразделению, выпускающему ПО.

В результате в графике разработки ПО много сотен позиций гораздо больше, чем этапов жизненного цикла, так как кроме разработки структурных единиц ПО, его сборки и квалификационных испытаний планируется выпуск отчетной документации, эксплуатационной документации, моменты обмена разработчиками проектной информацией о параметрах и порядке взаимодействия разрабатываемых структурных единиц ПО и т.п [ ].

Такой детальный подход позволяет убедиться в том, что каждая работа плана имеет исполнителя и что нет забытых работ. При контроле за ходом разработки ПО эти подробности не позволяют исполнителям отчитываться за проделанную работу «в целом и в общем». Подробный план позволяет избежать «изобретательности недобросовестных исполнителей», требуя конкретных результатов по конкретным работам.

В результате план разработки ПО СТС состоит из многих сотен событий. Его составление и согласование с исполнителями – серьёзная работа. Зато такой план выполняет свою координирующую и

контролирующую роль. Обычно применяются две формы представления плана : в виде сетевого графика и в виде диаграмм Ганта.

Сетевые модели программного проекта определяют связи между различными работами, представленными в виде событий графика, имеющих определённый номер. События графика связаны между собой стрелками, определяющими последовательность проведения работ. Каждое событие имеет срок завершения, а каждая операция – исполнителя (смотри рис.12).

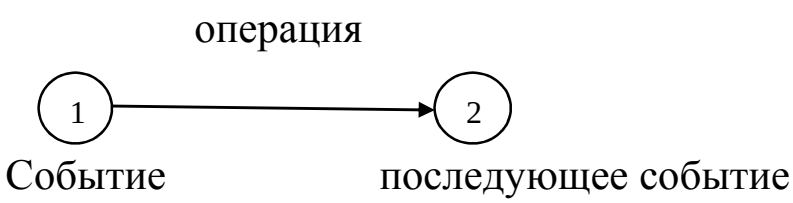


Рис.12. Изображение операции на сетевом графике

Сетевой график всегда начинается с единственного начального события и заканчивается событием завершения работ. Поскольку при планировании проекта многие операции выполняются параллельно, то между начальным и конечным событиями на графе плана может существовать несколько путей, каждый из которых будет иметь различную календарную длительность. Наиболее длинный из них называется критическим путём графика. Критический путь графика определяет срок выполнения проекта в целом. Критический путь графика всегда подвергаются анализу на предмет возможностей его сокращения. Если такая возможность находится, то образуется новый критический путь, который также подвергается анализу. Так происходит оптимизация сетевого графика проведения работ.

Если топология плана (события и связи между ними) и исполнители его – объективная часть плана, то плановые сроки выполнений событий плана носят более субъективный характер, что отражает объективное желание руководителей проекта завершить работу в требуемые жизнью сроки, как правило минимальные. С таким порядком вещей вполне можно мириться, если у того же руководства проектом правильные взгляды на коррекцию плановых сроков при необходимости.

Надо отметить, что планируются сроки создания ПО, которого еще нет. Какие сроки проведения работ надо поставить на каждую

из сотен позиций графика? Как доказать себе и руководителю, который традиционно выжимает минимальные сроки, обоснованность ваших предложений.

Здесь на помощь приходят оценки размера ПО и трудоемкости его разработки, полученные с помощью статистики: метод функциональных точек, модифицированных метрик Холстэда [3,4,11], а также опыт руководителя. Большую доказательную роль здесь имеет правильная **топология графика разработки, которая должна учитывать все необходимые работы.** Топология графика – графическое отображение связей между событиями плана, определяющее возможность последовательного или параллельного проведения работ. Это – объективная часть плана.

Правильная и подробная топология плана разработки ПО гарантирует от ситуаций типа «всё **в основном** есть», но работу завершить нельзя – упустили разработку одной программы и теперь все будут ждать пока она подтянется и встанет на сборку ПО. Правильный и подробный сетевой график позволяет также «расшить» узкие места путем изменения топологии.

Сетевой график очень удобная форма представления плана, на которой видны критические пути, объективные связи и зависимости событий плана друг от друга. На рис.13 приведён укрупнённый сетевой график разработки ПО СТС, а в таблице 2 приведена расшифровка операций этого графика.

Сроки графика разработки ПО и вопросы их коррекции

Сроки плана разработки часто несут субъективную - директивную составляющую. Разработка ПО, так же как и **разработка сложных систем вообще, не очень предсказуемый и детерминированный процесс.** Множество непредвиденных факторов и событий могут повлиять на ход разработки. Менеджеры ПО очень часто обещают характеристики и отличительные преимущества ПО, сроки разработки и требуемые деньги объективно не очень поддающиеся точным предсказаниям. Делают они это потому, что хотят себя «отделить» от конкурентов, а иногда под давлением руководства проектом или под давлением обстоятельств.

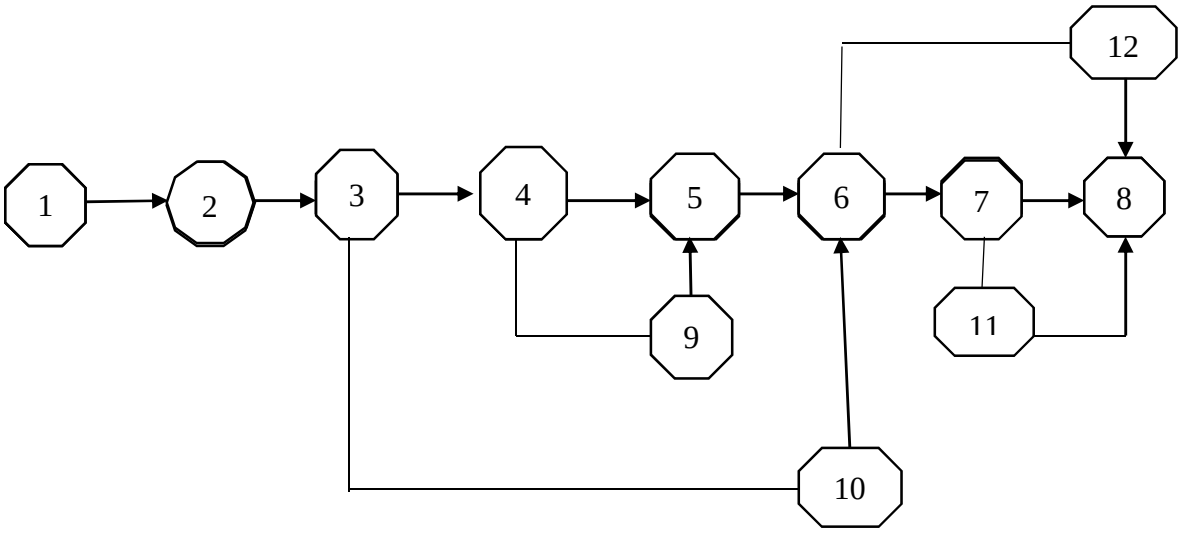


Рис.13 Упрощенный график разработки ПО СТС

Поэтому сроки топологически правильного графика работ не всегда выполняются. Причем это явление распространено и в России, и на Востоке, и на Западе. Здесь руководителям программных проектов важно держать руку «на пульсе» и своевременно предпринимать меры по усилению тех или иных сорванных направлений работ с коррекцией сроков графика.

Таблица2. Расшифровка позиций графика разработки ПО

События	**Операции**
1,2	Разработка требований к ПО
2,3	Проектирование ПО. Разработка структуры ПО
3,4	Проектирование алгоритмов частей структуры ПО
4,5	Кодирование ПО
5,6	Автономная отладка частей ПО(АО)
6,7	Комплексная отладка ПО(КО)
7,8	Подготовка отчёта и заключения по КО

4,9,5	Разработка тестов для АО
3,10,6	Разработка моделей внешней среды для КО
7,11,8	Разработка и выпуск ЭТД на ПО
6,12,8	Подготовка отчётов и заключений на АО

Свою лепту в установку недостаточно обоснованных сроков графика разработки ПО вносят и разработчики ПО. В программостроении уже стало обычным явлением то, что разработчики без достаточного основания называют слишком оптимистичные сроки, не учитывая множество препятствий, носящих в том числе и случайный характер. Иногда в этом сказывается желание понравиться заказчику и отделить себя от более осторожных конкурентов. Среди опытных руководителей и заказчиков даже распространено неписаное правило: умножать на 2 оценку трудоемкости, которую сделал разработчик программы.

Ещё один распространенный источник занижения сроков у разработчика – необоснованные ожидания от применения новых технологий и средств разработки. Эти ожидания, как правило, не оправдываются. Согласно статистике средняя производительность в производстве программ растет всего лишь на 3–5% в год.

С другой стороны, часто появляется «агрессивный» график программного проекта с весьма сжатыми и нереалистичными сроками. Такой график появляется из-за того, что руководство проектом и/или заказчик боятся «отпустить проект», полагая, что согласно закону Паркинсона, работы по проекту займут все отведенное для него время. «Студенческий синдром» является оправданием подобных действий: если разработчикам будет выделено слишком много времени, то в начале они будут работать спустя рукава, а затем аврально с ошибками наверстывать упущенное время. Следствием подобных опасений является, как правило, директивное занижение плановых сроков реализации проекта.

Если график излишне жесткий, то с целью сэкономить время, недостаточно времени и внимания уделяется начальным этапам - анализу требований и проектированию. Исправление ошибок, допущенных на этих этапах, приведет к существенным дополнительным затратам впоследствии.

При этом важно понимание, что работать без плана в таком сложном деле, как разработка ПО, нельзя. **Ситуация не выходит из**

под контроля тогда, когда график работ пусть и с не очень устраивающими жесткими сроками имеется. Тогда участники разработки скоординированы и могут пытаться его выполнить. Если же выполнить график не удаётся, то его надо корректировать, в противном случае работы проводятся вне графика т.е. не координируются и не управляются.

Рассматривая возможности компенсации нарушений графика, следует иметь в виду следующие альтернативные варианты действий:

-уменьшить (упростить) требования к ПО (здесь можно попытаться согласовать у заказчика этапность выполнения полных требований путем выпуска нескольких версий продукта),

-добавить людей,

-добавить деньги для стимулирования,

-пересмотреть сроки.

Чаще всего используется комбинация этих действий.

Диаграммы Ганта.

Еще одной формой представления плана являются диаграммы Ганта. В отличие от сетевого графика диаграммы Ганта – это матрица. На диаграмме Ганта по вертикали записывается перечень последовательных работ, а по горизонтали – календарная сетка. В этих координатах для каждой работы указывается ее длительность. Это – достаточно древний способ представления плана, по которому удобно контролировать планируемое и реальное состояние работы.

Однако для сложных работ сетевой график, как цепочка последовательных и параллельных событий, связанных между собой, показывает в графическом виде «что от чего зависит», что не видно на диаграммах Ганта. Поэтому пересчитывать критические пути и искать выходы из критических ситуаций планирования удобнее на сетевом графике, а контролировать ход работ удобнее на диаграммах Ганта.

Часто для представления генеральных планов создания СТС в целом руководителям более высокого ранга, план разработки ПО обобщается до нескольких десятков событий. Это вполне допустимо и практикуется повсеместно, но разрабатывать ПО по такому «обрезу» нельзя. Таким образом, на практике существует иерархия планов в зависимости от конкретного предназначения каждого из них.

Глава 7. Управление рисками программного проекта

Риски - нештатные ситуации процесса разработки

В любой инженерной проектной разработке рассчитывать на то что все пойдет гладко нельзя. Это касается как техники дела, так и вопросов планирования и управления проектом. **Исследования программных проектов**, окончившихся неудачей (их по статистике 35% от общего количества), показали самые распространёнными причины провалов, которые мы ранее рассмотрели и которые ниже мы ещё раз перечислим в виде перечня рисков.

В первой лекции мы уже говорили о том, что программная инженерия остается и, в ближайшем будущем, будет оставаться производственной дисциплиной с высоким уровнем рисков. Если задуматься, то все, что мы делаем, управляя проектом разработки ПО, направлено на борьбу с рисками. Мы управляем программным проектом, опасаясь не уложиться в срок, перерасходовать ресурсы, разработать не тот продукт, который требуется, разработать ПО низкого качества.

Работа по обеспечению безопасности начинается с определения рисков и угроз. Управление техническими системами предусматривает создание ряда мероприятий по управлению в нештатных ситуациях: их обнаружению, локализации, восстановлению работоспособности и правильной информации системы. Все это делается с целью уменьшения ущерба при отказах и направлено на продолжение функционирования системы пусть и с возможным ухудшением показателей качества функционирования.

Перечень подобных нештатных ситуаций должен быть подготовлен заблаговременно на этапе проектирования системы также, как и упомянутые мероприятия по их парированию и только в этом случае проявившаяся нештатная ситуация будет встречена системой во «все оружии», быстро обнаружена и парирована и ущерб от неё будет действительно минимальным.

Перечень нештатных ситуаций и методов их парирования отражается в политике безопасности – совокупности документированных решений, разработанных с целью защиты информации и ассоциированных с нею ресурсов систем.

Точно также при управлении разработкой программных проектов должна быть разработана политика безопасности – должны быть сформулированы и оценены нештатные ситуации процесса разработки ПО – риски, а также предусмотрены заблаговременно варианты действий, направленных на минимизацию ущерба от этих отклонений от запланированного хода разработки ПО.

Определение риска:

Риск- случайное событие или условие, наступление которого может отрицательно или положительно сказаться на целях проекта. Риск – это проблема, которая еще не возникла, а проблема – это риск, который материализовался. Риск характеризуется вероятностью наступления и последствиями воздействия на проект.

Риск определяется:

1. Причиной или источником риска - явлением, обстоятельством обусловливающее наступление риска.
2. Симптомами риска, указания на то, что событие риска произошло или вот- вот произойдёт.
3. Последствиями реализации риска. Проблема или возможность, которая может реализоваться в проекте в результате исполнившегося риска.
4. Влиянием риска. Влияние реализовавшегося риска – более или менее отдаленный прогноз на возможность достижения целей проекта.

Воздействие риска обычно касается стоимости, графика и технических характеристик разрабатываемого продукта. **При управлении рисками приходится рассматривать не только его влияние на проект, но и учитывать вероятность наступления.** Например, всегда есть вероятность того, что метеорит упадет на офис центра программных разработок, и это будет иметь катастрофические последствия для проекта. Однако, вероятность наступления этого события настолько мала, что мы в большинстве проектов **принимаем это риск** и не пытаемся им управлять.

Отчего возникают риски

Принято выделять две категории рисков:

1.Известные «неизвестные – возможные случайные события, влияющие на выполнения плана**».** Это те риски, которые можно идентифицировать и подвергнуть анализу в процессе разработки

ПО. В отношении таких рисков можно спланировать соответствующие защитные действия.

2.Неизвестные «неизвестные– возможные случайные события, влияющие на выполнения плана**».** Риски, которые невозможно идентифицировать при разработке ПО и, следовательно, спланировать ответные действия. Неизвестные риски – это непредвиденные обстоятельства. Единственное, что мы можем в этом случае предпринять, это, например, **создать управленческий резерв бюджета** проекта на случай непредвиденных, но потенциально возможных изменений. Управленческие резервы на непредвиденные обстоятельства не входят в базовый план по стоимости проекта и не распределяются по этапам проекта, как бюджет, но включаются в бюджет проекта. На расходование этого резерва менеджер проекта, как правило, обязан получать указание вышестоящего руководства**.** Рассмотрим отчего возникают известные риски.

Риски, связанные с требованиями. Процесс разработки программного обеспечения начинается с определения требований и вариантов использования системы. Обычно они формализуются в том или ином виде технического задания группе разработчиков. Основная проблема заключается в том, что некоторые ключевые требования, которые требуются для реализации системы могут быть пропущены, поскольку заказчики могут посчитать их настолько очевидными, что их даже не нужно упоминать, но для разработчика они не очевидны. Другие требования могут быть не так поняты разработчиками. В итоге создаваемая программная система будет выполнять не то, что хотели пользователи или заказчик.

Не редки случаи, когда разработчик ПО под давлением или по недомыслию принимает требования, которые он выполнить не в состоянии. Иногда это делается с целью перехватить заказ с надеждой последующей коррекции требований. Это удаётся не всегда.

Технологические риски. Эта группа рисков объединяет риски, связанные с используемыми технологиями. Со времён зарождения отрасли производства программного обеспечения было создано великое множество технологий. Технологии создавались для того, что бы решать задачи по некоторому шаблону: не всегда «с чистого листа - всё заново и как хочется», а с уже известными выделенными этапами и средствами, доказавшими свою практическую ценность. Использование новых технологий всегда таит в себе подводные камни.

Риски, связанные с персоналом и его квалификацией. Насколько стабильна команда разработчиков? Насколько сотрудники, которые участвуют в проекте, опытны в применяемых технологиях? Эти вопросы возникают при управлении проектом и являются постоянной заботой руководителя. Есть и другие риски, связанные с персоналом. Сотрудники компаний, руководители больших и малых подразделений часто ведут тонкую политическую игру, целью которой может быть продвижение по служебной лестнице и получение максимальной возможной власти. Эти цели весьма далеки от целей конкретного проекта, поэтому тихое противодействие отдельных сотрудников или неформальных групп может свести на нет все усилия менеджера проекта.

Планирование управления рисками. Идентификация рисков. Допущения проекта. Методы реагирования на риски

Цели управления рисками проекта – снижение вероятности возникновения и/или значимости воздействия (ущерба) неблагоприятных для проекта событий. Можно провести аналогию с управлением сложными техническими системами, для ряда которых обязательным условием является наличие перечня нештатных ситуаций и управление выходом из них, минимизирующее ущерб. Управление рисками процессов разработки программного обеспечения является средством минимизации вероятного ущерба от нештатных ситуаций процессов разработки невыполнения требований по качеству, бюджету и срокам разработки.

Адекватное управление рисками в компании – признак зрелости производственных процессов. Отказываться от управления проектными рисками это всё равно, что в кинотеатре не иметь огнетушителей и плана эвакуации на случай пожара. Хотя риск может воздействовать и на сроки проекта, и на качество получаемого продукта, но все эти отклонения могут быть оценены в размере ущерба в денежном эквиваленте. Например, последствия задержки по срокам для заказной разработки может быть выражена в сумме денежных санкций, определенных в контракте.

Идентификация рисков – это выявление рисков, способных повлиять на проект, и документальное оформление их характеристик. Это итеративный процесс, который периодически повторяется на

всем протяжении проекта, поскольку в рамках его жизненного цикла могут обнаруживаться новые риски.

Для определения рисков программного проекта можно использовать различные доступные контрольные списки рисков проектов разработки ПО, которые следует проанализировать на применимость к данному конкретному проекту.

Каждый проект задумывается и разрабатывается на основании ряда гипотез, сценариев и допущений. Как правило, в описании требований к проекту перечисляются принятые допущения - факторы, которые для целей планирования считаются верными, но определенными без привлечения достаточных доказательств**. Неопределенности, сформулированные в допущениях проекта, следует обязательно рассматривать в качестве потенциальных источника возникновения рисков проекта.** Анализ допущений позволяет идентифицировать риски проекта, происходящие от неточности, несовместимости или неполноты условий выполнения проекта.

Неправильно думать, что планирование управления рисками заканчивается после составлении перечня рисков. Это всего лишь стартовая точка. Реализация ответных действий на каждый реализовавшийся риск является тем, что на самом деле добавит ценности проекту и минимизирует негативные эффекты. Действия – вот что реально уменьшает ущерб. Планирование реагирования на риски – это процесс разработки путей и определения действий по снижению угроз для проекта. Данный процесс начинается после проведения качественного и количественного анализа рисков.

Рассмотрим подробнее четыре метода реагирования на риски:

1. **Уклонение от риска**. (избегание риска). Избегание означает, что вы стараетесь организовать процесс разработки ПО таким образом, чтобы в дальнейшем не встретить этот риск. Некоторые риски, возникающие на ранних стадиях проекта, можно избежать при помощи уточнения требований, получения дополнительной информации или проведения экспертизы. Например, уклониться от риска можно, если отказаться от реализации рискованного функционального требования или самостоятельно разработать необходимый программный компонент, вместо ожидания поставок продукта от субподрядчика, для которого существует вероятность не поставки в требуемый срок. В первом случае вопрос должен быть согласован с заказчиком.

2. Передача риска подразумевает переложение негативных последствий угрозы с ответственностью за реагирование на риск на третью сторону. Передача риска просто переносит ответственность за его управление другой стороне, но **риск при этом никуда не девается.** Передача риска практически всегда предполагает выплату премии за риск стороне, принимающей на себя риск. Например, оговаривается в контракте заказ на стороне разработки рискованного компонента по фиксированной цене. В IT часто приходится формулировать риски в виде допущений, тем самым передавая его заказчику. Например, оценивая проект внедрения, мы можем записать допущение о том, что производитель не изменит стоимость лицензий на базовое ПО. Если изменение стоимости все же произойдет, то заказчик оплатит разницу, поскольку с этим риском – допущением он согласился. Ну и главный прием передачи риска – страхование риска в специализированных организациях.

3. Снижение рисков предполагает понижение вероятности или последствий негативного рискованного события до приемлемых пределов. Принятие предупредительных мер по снижению вероятности наступления риска или его последствий часто оказываются более эффективными, нежели усилия по устранению негативных последствий, предпринимаемые после наступления события риска. Например, регулярная ревизия поставок заказчиком может снизить вероятность риска его неудовлетворенности конечным результатом. Если в проектной команде высока вероятность увольнения сотрудников, то введение на начальной стадии в проект дополнительных (избыточных) людских ресурсов снижает потери при увольнении членов команды, поскольку не будет затрат на «въезд» в проектный контекст новых участников.

4. П**ринятие риска** означает, что команда проекта осознанно приняла решение не изменять план управления проектом в связи с риском или не нашла подходящей стратегии реагирования. Мы вынуждены принимать все «непредвиденные риски».

Принятие риска –это то, что всегда происходит, когда мы вообще не управляем рисками. Если же мы управляем рисками, то мы можем страховать риски, закладывая резерв в оценки срока завершения проекта или трудозатрат на него. Активное отношение к принятым рискам может состоять в разработке план реагирования на риски.

Наиболее распространенные риски программных проектов

По технической литературе кочуют длинные списки рисков программных проектов. Помятуя, что психологами доказано, что для успешного анализа связей между объектами число их не должно превышать число Миллера 7 ± 2, рассмотрим **мой список** из 7 главных рисков программных проектов.

1. Требования заказчика отсутствуют или не полны , или подвержены частым изменениям. К этому можно добавить только одно: «Когда человек не знает, к какой пристани он держит путь, для него никакой ветер не будет попутным» (Сенека Луций Аней, философ, 65г. до н.э.)

2. Требования заказчика трудно выполнимы или вообще не выполнимы.

3. Отсутствие необходимых ресурсов и опыта у разработчика.

4. Неполнота планирования. «Забытые работы» и неправильная топология графика.

5. Плохое управление изменениями,

6. Плохая верификация (в том числе при согласование проектных документов).

7. Плохая и недокументированная отладка, которая сводит на нет прекрасные замыслы проекта

При планировании управления рисками часто возникают ситуации, когда решение по методу реагирования на риски зависит от позиции заказчика. Если с заказчиком не удается найти взаимоприемлемое решение при **первоначальной оценке проекта, то разумно попытаться договориться о выполнении проекта в 2 этапа с самостоятельным финансированием каждого этапа**. С вменяемыми заказчиками это часто удается. Прием с разбиением проекта на этапы часто оказывается эффективным при коррекции сорванного графика разработки ПО.

Характеристики процессов контроля, принципы контроля

Задачи контроля состоят в том, чтобы обнаружить соответствия фактических результатов деятельности организации её целям и задачам. Регулярный контроль создает обратную связь в управлении и

позволяет своевременно обнаружить отклонения фактических результатов от запланированных и своевременно внести коррективы в управление.

Процессы контроля состоят из процессов сбора контрольной информации, её обработки, сравнения с плановой и выделение отклонений. С целью повышения достоверности контроля необходимо иметь несколько источников контрольной информации, каждый из которых необходимо рассматривать и учитывать в сравнении.

Процессы контроля характеризуются:

1 регулярностью,

2.достоверностью,

3.запаздыванием,

4.степенью охвата.

С точки зрения регулярности контроль может быть непрерывный (дискретный с допустимым периодом) и контроль может быть эпизодическим. Последнее – плохо.

Контроль с точки зрения степени охвата может быть сплошным, а также выборочным. Правильная тактика в этом вопросе с учетом затрат времени и средств на осуществление контроля периодически проводить выборочный контроль, а с гораздо большим периодом – сплошной

Основные принципы контроля:

1.Нацеленность на поиск и предотвращение ошибок или срывов планов, а не на демонстрацию их отсутствия.

2.Не практиковать контроль «по случаю», который не может дать полной объективной картины состояния процесса разработки и его плана.

3.Проводить контроль с минимальным запаздыванием. Контроль вдогонку не позволяет вводить корректирующие воздействия, а будет констатировать только срыв.

4. Не проводить тотальный постоянный контроль действий персонала, а не плана, так как он будет приводить к несамостоятельности персонала.

Реализацию данных принципов контроля необходимо сопровождать практикой ежедневного оперативного контроля хода работ с ответом на четыре вопроса:

1. Что сделано вчера?

2. Что надо сделать сегодня?

3. Какие имеются организационные и технические трудности?

4. Какова оценка действительного хода разработки по сравнению со сроками графика**?**

Контроль за качеством разработки ПО связан с анализом полученных ошибок и оценкой их влияния на дальнейший процесс разработки. В процессе разработки ПО могут возникнуть проблемы, приводящие к дефектам конечного продукта ПО. Разрешение этих проблем, а попросту ошибок – заблуждений алгоритмического характера может потребовать исправления не одной, а ряда программ ПО.

По данным проблемам под руководством подразделений, отвечающих за разработку ПО выпускаются обзоры - документы, в которых отражаются:

наличие проблемы,

определение источника проблемы,

точную локализацию дефекта ПО,

коррекцию требований к ПО (при необходимости),

порядок и сроки исправления ПО, объем необходимой отладки,

порядок и сроки исправления ПО на заделе систем (поставленных систем) в том числе и на поставленной эксплуатационной документации.

Подобный обзор можно назвать «Техническим решением проблемы …ПО». Иногда он определяет очередную итерацию в разработке ПО. Для этой цели необходимо планировать обзоры с состоянием дел.

Объективно оценить и проконтролировать можно только то, что можно измерить. Для объективной оценки необходимо вводить метрики проекта – количественные показатели оценки различных характеристик проекта и процесса его выполнения. Метрики могут вводиться как для всего проекта в целом, так и для отдельных видов работ. Общими метриками проекта являются:

1.Количество выполненных фаз / действий / работ.

2.Продолжительность каждой работы и её соответствие плановым срокам.

3.Степень загрузки ресурсов и исполнителей на отдельных этапах.

4.Количество изменений в проекте.

Оперативный план проекта в виде сетевого графика или диаграммы Ганта является мощным информационным инструментом

контроля за ходом работы над проектом. Для этого достаточно помечать выполненные позиции и тогда картина, что запланировано на данный момент и что фактически сделано становится ясной и наглядной.

В этой связи сбор объективной информации о проделанной работе необходимо организовать. Обычно это делается распределенным образом, когда сами исполнители тех или иных работ отчитываются об их завершении.

Нет документального подтверждения или письменного отчета от исполнителя работы – значит работа не выполнена. Если срок прошел, а работа не выполнена или не перенесена установленным порядком у руководителя, то план подразделения – исполнителя считается не выполненным и исполнители данных работ депремируются или являются кандидатами на объявление взыскания вплоть до отстранения от должности. Конечно, такие суровые меры должны применяться не сразу, а в соответствии с законодательством.

Отчеты о выполнении событий плана можно вносить в электронном виде в систему планирования и контроля с подтверждением выпуска соответствующих артефактов (документов), положенных по завершению работ закрываемого пункта, но чаще всего в больших организациях имеются специальные службы (подразделения) координации работ, которые сбором и представлением этой контрольной информацией занимаются, используя ПО автоматизации управления проектами.

Для успешной работы таких систем все важные события плана должны быть наблюдаемы и сопровождаться оформлением завершенной работы для того, чтобы контроль был объективным. В отличии от строящегося корабля, самолета, телевизора, когда руководители могут зримо наблюдать состояние производства, разработка ПО не видима и происходит в головах и файлах разработчика, недоступных для обозрения администратором без специальных средств визуализации.

Средством визуализации хода разработки являются **отчетные документы, выпуск которых обязателен** по завершению этапа или в специально придуманных контрольных точках**, имеющихся в графике разработки.** Это - отчеты об отладке, заключения по верификации, специально оформленные и представляемые файлы.

Глава 8. Управление проектом и лидерство. Работа руководителя.

Лидерство и формально правовая власть

Работа руководителя программного проекта связана прежде всего с управлением людьми и должна быть направлена на решение проблем разработки. Это управление формально опирается в общем случает на права и обязанности руководителя, закреплённые в законодательстве. В конкретных случаях права и обязанности руководителя закреплены в должностных положениях и инструкциях. Но кроме этой формальной власти во многих рабочих коллективах существует реальная власть неформального лидера.

Реальная власть - это власть личности и её авторитета. Если она основана на знании и опыте личности, на её способностях обрабатывать информацию с высокой скоростью, то это позволяет решать задачи, которые не под силу другим членам коллектива. Порожденная таким образом власть, основанная на компетенции, способствует определению путей решения проблем. Власть личности также может опираться на присущую ей харизму. Харизма это дарованная человеку судьбой способность и умение обаять и подчинять своему обаянию людей. Харизмой обладали Христос, Магомет, Наполеон, ряд политических деятелей и полководцев. Эта власть не прочна, так как основывается на эмоциях людей. Она определяется также готовностью человека подчиниться ей, но симпатии людей меняются.

Формально правовая власть- власть по закону опирается на законодательные нормы. Законом поддерживается власть собственника.(государство, частное лицо) но собственник редко управляет сам и нанимает управленца, отдавая в его распоряжение ресурсы материальные и информационные.

Границы реальной и формальной власти часто не совпадают. Часто их носители – различные люди. Поэтому первый шаг руководителя при создании эффективной команды – это попытаться стать не только формальным, но и не формальным лидером, вокруг которого сможет сплотиться рабочий коллектив.

Лидера нельзя назначить. Лидерство, в первую очередь, это умение управлять своей собственной жизнью и только потом другими

людьми. Сверхвысокое значение коэффициента интеллекта IQ, к сожалению, в этом не поможет. Личная эффективность человека на 80% определяется его коэффициентом эмоционального интеллекта EQ (Emotional Intelligence) – способностью понимать и эффективно взаимодействовать с другими людьми. В отличие от IQ, который формируется в ранней молодости и затем практически не меняется, EQ можно повышать на протяжении всей жизни. Если, конечно, прилагать к этому усилия.

Опросы общественного мнения, проведенные в нашей стране показали, что больше всего ценят подчиненные в руководителях:

- справедливость, честность, порядочность – 76% опрошенных,
- компетентность, умение организовать работу- 24% опрошенных,
- понимание жизненных проблем работников – 25% опрошенных,
- требовательность – 4% опрошенных.

Отсюда ясно каким образом, управляющий может получить признание команды – стать её лидером. Для того этого необходимо:

- Полное доверие команды к действиям и решениям лидера, признание его человеческих качеств, убежденность в его честности, порядочности, вера в его искренность и добросовестность.
- Признание командой профессиональной компетентности и превосходства лидера.

Если руководитель не смог стать лидером, он будет вынужден применять в своей практике управленческие антипаттерны. Для такого руководителя характерны чрезмерная настороженность, скрытность, неспособность делегировать полномочия, постановка во главу угла при решении всех вопросов сохранение своей должности. Всё это очень мешает эффективной работе. Вместо мотивирования сотрудников на успех, применение антипаттернов мотивирует их на избежание риска и негативных для себя последствий, подавляет свободу, самостоятельность, творчество и инициативу. Это приводит к деструктивному подчинению, когда все работают строго по инструкции и только в соответствие с указаниями руководства.

Наконец, применение антипаттернов - это стрессы, усталость участников, личные проблемы, увольнение наиболее профессиональных сотрудников и провал проекта.

Ещё качества плохого руководителя:

- уверенность в своей непогрешимости,

- склонность к длинным речам - отсутствие чувства меры,

- обилие оправданий непредвиденными обстоятельствами, которые прикрывают неумение предвидеть последствия своих действий.

Компетенции эффективного руководителя. Стратегии руководства

Эффективный руководитель для управления программными проектами обязан обладать следующими компетенциями:

1. Видением целей, перспектив и стратегии их достижения.

2. Глубоким анализом проблем разработки и поиском новых возможностей(профессионал)

3. Способностью найти выход из сложной ситуации

4. Нацеленностью на успех, стремление получить наилучшие результаты.

5. Навыками в разрешении конфликтов.

6. Терпимостью, умением принимать людей такими, какие они есть, принятие их

права на собственное мнение и на ошибку.

7.Умением взять ответственность на себя.

8 Умением организовать исполнительство и деловые коммуникации - переписку

9. Умением организовать работу по плану, осуществлять контроль и координацию работ,

10. Способностью активно "обеспечивать", "доставать", "выбивать" и т.д.

Не существует одной лучшей стратегии руководства. В зависимости от готовности участников рабочей группы выполнять задания руководителя и состояния программного проекта руководитель может использовать одну из 4-х стратегий :

1. «Директивное или авторитарное управление». Руководитель говорит, указывает, направляет, устанавливает. Жесткое назначение работ, строгий контроль сроков и результатов.

2. «Объяснения или демократическое управление». Руководитель объясняет, проясняет, убеждает. Сочетание директивного и коллективного управления. Объяснение руководителем своих решений.

3. «Участие или либеральное управление». Руководитель участвует, поощряет, сотрудничает, проявляет преданность. Приоритетным является коллективное принятие решений, обмен идеями, поддержка инициативы подчиненных.

4. «Делегирование». Лидер делегирует полномочия, наблюдает, обслуживает. «Не мешать» - главный принцип пассивного управление такого сформировавшегося лидера.

Выдающиеся руководители владеют всей палитрой стилей руководства и умело ими оперируют, в нужный момент «натягивая вожжи», а в нужный момент их ослабляя.

Вас назначили руководителем в новый коллектив:

1. Вы еще не получили признания, а дело делать надо. Стратегия: «Директивное управление».

2.Все знают о ваших прежних сложных и успешных проектах. Все признают ваше превосходство, но доверия к вам нет, так как никто не знает, какой ценой были достигнуты ваши победы. Стратегия: «Участие».

3. Вы сами недавно были участником команды. Вас назначили руководителем этой команды. Доверие есть, а уверенности в правильности ваших действий нет. Стратегия: «Объяснения».

4. Между вами и участниками команды быстро установлено взаимное доверие. Все достаточно мотивированы на успех проекта. Стратегия: «Делегирование».

Основные усилия руководителя, если он стремится получить наивысшую производительность рабочей группы, должны быть направлены на изучение и изменение объекта управления: людей и их взаимодействия.

Поэтому эффективный руководитель является оптимистом, при этом твердо знает, что окружающий мир несовершенен; воспринимает каждую новую проблему, как дополнительную возможность подтвердить собственный профессионализм в своих глазах и во мнении коллег.

Управление персоналом, мотивация

В процессе разработки ПО роль человеческого фактора очень велика. Люди, умеющие разрабатывать эффективное и надежное ПО, являются самым ценным активом фирм, разрабатывающих сложные

технические и информационные системы, поскольку ПО определяет концептуальный уровень этих систем.

Имеются четыре необходимых и достаточных условия для того, чтобы сотрудник эффективно решил поставленную задачу. Это

1. Понимание целей работы.
2. Умение ее делать.
3. Возможность ее сделать.
4. Желание ее сделать.

Для того чтобы обеспечить выполнение этих условий, руководитель должен уметь эффективно выполнять четыре функции:

1. Направлять. Если сотрудник не понимает что делать, задача руководителя - обеспечить общее видение целей и стратегии их достижения.

2. Обучать. Если сотрудник не умеет, задача руководителя – «обучать», быть наставником и образцом для подражания.

3. Помогать. Если у сотрудника не получается выполнить работу, задача руководителя – «помогать», обеспечить его всем необходимым, убрать препятствия с его пути.

4. Вдохновлять. Если у сотрудника не достаточно желание выполнить работу, задача руководителя – «вдохновить», обеспечить адекватную мотивацию участника на протяжении всего проекта.

Менеджеры проектов ПО должны уметь мотивировать действия разработчиков .

Программист состоит из четырех «компонентов»: тело, сердце, разум и душа.

1. Телу необходимы деньги и безопасность.
2. Сердцу - любовь и признание.
3. Разуму – развитие и самосовершенствование.
4. Душе – самореализация и внешнее признание достижений.

Если все это предоставить сотрудникам, то эффективность их труда возрастет многократно. Обычно данные потребности излагаются в виде пирамиды Маслоу

Первые две базовые потребности физиологические и безопасности для разработчиков ПО обычно выполняются, Социальные потребности удовлетворяются предоставлением возможности общения внутри группы, занятием определенной роли в группе. Потреб-

ности в самореализации удовлетворяются предоставлением определенного уровня ответственности и самостоятельности. Потребности в оценке удовлетворяются открытым признанием достижений.

Любой человек на производстве принадлежит к одному из трех типов.

Первый тип – лидер. Это человек, который стремится управлять другими людьми, проектами, и для которых невозможно быть просто «винтиком» в рабочем механизме.

Второй тип – технарь. Это человек, который получает удовольствие от процесса поиска решения, от решения задачи. Для него не столь важно быть начальником.

Третий тип назовем «общительный». Эти люди разносят информацию и большую часть времени занимаются межличностной коммуникацией, в лучшем случае популяризуя результаты других людей.

Команда из одних лидеров не даст хороших результатов- будет постоянная борьба за власть.

Команда из одних технарей не будет соблюдать сроки, бюджет, увлекаясь поиском наилучшего решения задачи.

Команда из одних «общительных» будет организовывать самые лучшие праздники и корпоративы, но работа спориться не будет.

При подборе персонала должны приниматься во внимание баланс лидеров, технарей и «общительных», а также следующие качества, которые мы ранжируем следующим образом: знания в предметной области, знания в области программной инженерии в том числе технологии разработки ПО,

знание программирования,

способности к адаптации.

Конфликт и управление проектом в этих условиях

Конфликт – столкновение противоречивых интересов, позиций, мнений, взглядов и идей всегда проявляются в человеческих коллективах. Люди стремятся разрешить конфликт в свою пользу с помощью убеждения или силовым давлением. В конфликтной ситуации возможно также достижение компромисса или достижения консенсуса.

Конфликты часто имеют положительный эффект и предшествуют позитивным преобразованиям, так как выявляют скрытые

противоречия, «подводные камни», разнообразные точки зрения – новое рождается через конфликт.

Но часто конфликт носит деструктивный характер и разрушает систему взаимоотношений в организации.

Основные причины конфликтов:

1.борьба за ограниченные ресурсы,

2.неправильное распределение функций управления, недоговоренностях в распределении обязанностей и ответственности,

3.несходства точек зрения,

4. наличие конфликтных личностей, неспособных к компромиссам и убеждению

Конфликты часто приводят к образованию клик- формированию групп сотрудников с целью захвата власти, противостояния действиям противостоящей группы или линии руководства. Конфликты часто сопровождаются интригой – нечестным запутыванием окружающих для вовлечения их в деятельность в интересах интригующих

.При разрешении и предотвращении конфликтов велика роль руководителя. Пожар легче сразу потушить, чем дать ему возможность разгореться.

Основные действия по предотвращению конфликтов:

-улучшение условий труда,

-справедливое и гласное распределение ресурсов,

-открытое и без недомолвок распределение работ и ответственности без наличия белых пятен и самотека в этих вопросах,

-контроль за соблюдением правил и традиций.

Основные управляющие действия по разрешению конфликтов:

- демонстрация невозможности достижения желаемого под давлением конфликта,

- организация совместного поиска путей примирения путем переговорного процесса,

-объединение всех общей целью,

-посредничество уважаемого обоими сторонами лица.

-применение методов административного воздействия.

Глава 9. Управление принятием решений при разработке ПО. Достижение компромисса и консенсуса. Обзор систем управления проектами

Коммуникации при управлении программными проектами

Основным фактором в разработке программного обеспечения является возможность коммуникации (общения участников проекта). Общение может проводиться в различных формах от строго формализованного (стандартизированная документация) до полностью неформализованного (вопрос-ответ соседу, обсуждение в неформальной обстановке).

На рис. 14 изображена некая кривая, иллюстрирующая эффективность различных способов общения. Кривая является обобщением ряда исследований в этой области. Видно, что эффективность общения падает по мере возрастания степени его формализованности.

С одной стороны, в этом нет ничего удивительного – старый принцип бюрократа гласит: хочешь получить отказ – пиши письмо, хочешь получить обещание – звони по телефону, хочешь добиться результата – езжай сам и общайся лично.

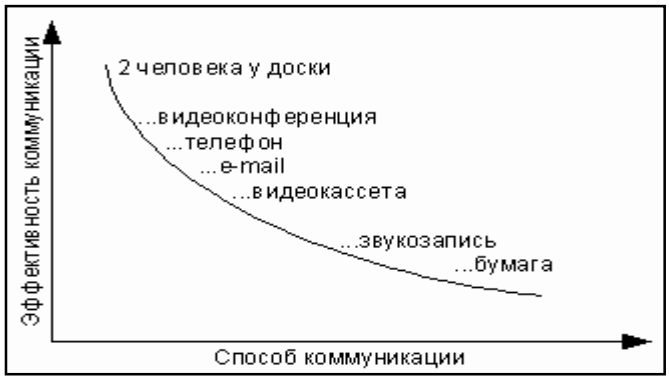


Рис.14 Эффективность процессов коммуникации

Но с другой стороны, надо помнить, что коммуникации в команде определяются количеством участников (рабочих связей): при двух участниках – это одна связь, при n участниках – n(n-2)/2. Т.е. неформальное, «живое» общение эффективно только в относительно небольших, хорошо организованных (сработавшихся) коллективах. Мы это отмечали, когда говорили об agile технологиях разработки ПО

Управление принятием решений при разработке ПО. Достижение компромисса и консенсуса

Целью общения в команде разработчиков являются обсуждение текущих проблем и вопросов и принятие решений. Как правило в процессе обсуждения высказываются различные мнения. Это хорошо. Конечно есть случаи, когда в процессе обсуждения люди бояться высказать своё мнение. Это плохо, поскольку в спорах рождается истина. Далее мы рассмотрим некоторые проблемы организации обсуждений и принятия решений. Начнем с принятия решений.

Итак, принятое в результате обсуждения решение может быть достигнуто в результате компромисса или в результате консенсуса. В чем разница этих результатов?

Начнем с определений :

Компромисс - соглашение, достигнутое посредством взаимных уступок. **Консенсус** (коллективное мнение) - общее для конкретной группы мнение.

В чем же разница?

Компромисс – это среднее решение, которое может оказаться (и, как правило, оказывается) хуже каждого из вариантов. Достигается путем взаимных уступок (мы согласимся с вашим вариантом интерфейса, если вы согласитесь с нашей организацией базы данных). Компромисс может быть принят большинством (голосованием).

Консенсус – это оптимальное решение, сочетающее лучшее из предложенных вариантов. Достигается путем обсуждения, анализа и генерации новых идей. Принимается общим согласием (все согласны, что найдено лучшее решение).

Приведем следующий пример компромисса и консенсуса. При обсуждении вопроса о размещении кнопок панели инструментов выдвинуты два варианта: горизонтально и вертикально. Компромисс – по диагонали (нелепое решение). Консенсус – настраиваемая пользователем панель (лучшее решение, включающее оба варианта на основе новой идеи – настраиваемая панель).

Как добиться консенсуса?

В отличие от компромисса, который чаще всего достигается в результате политических маневров и взаимных уступок, достижение консенсуса требует конструктивного и плодотворного напряжения всей команды и особого искусства управления командой. При этом рекомендуется придерживаться следующих принципов и правил:

1. Вера в достижение консенсуса – каждый член команды должен доверять другим в том, что обсуждение приведет к поиску оптимального решения, а не к победе **чьих то личностных мнений.** Создание такой атмосферы взаимного доверия является важнейшим в создании эффективной команды. Следует понимать, что взаимное доверие появляется не само по себе, а является результатом нескольких удачных консенсусов.

2.Участием всех в выработке и принятии оптимальных решений.

3.Созданием у каждого осознания причастности к принятым решениям.

4.На обсуждение люди должны приходить не со сформированной позицией и установкой ни шагу назад, а с вариантами возможных решений.

Объективность принимаемых решений связана с попыткой ограничить проявления чувств и эмоций при обсуждении вопросов. Чувства и эмоции являются неотъемлемым свойством человеческой природы. Избежать их полностью вряд ли удастся, но для приведения их «в норму» можно использовать следующие правила проведения обсуждений с целью поиска консенсуса для принятия решения:

1.Критерии оценки вариантов решения обсуждаемой проблемы – для объективности обсуждения крайне важно заранее договориться о критериях оценки – установить список критериев и выполнить их ранжировку по степени важности.

2.Разделение фактов и мнений. Факты – объективные показатели, выраженные в большинстве случаев количественно (но не обязательно): быстродействие, время отклика. Мнения – то, что не основано на фактах. Мнениями не следует пренебрегать, т.к. они часто основаны на опыте, интуиции.

3.Замена позиций – в случае, когда обсуждение всё же заходит в тупик, бывает полезно предложить участникам изменить точку зрения: «перечислите, пожалуйста, сильные стороны варианта Вашего оппонента и слабые стороны Вашего варианта»

4.Руководитель должен быть нейтрален. Руководитель (лидер) может принимать активное участие в обсуждении, но только на правах равного и поручить в этом случае руководство собранием другому человеку.

Роль руководителя в достижении консенсуса состоит в том, чтобы дать всем возможность высказаться и предложить свои варианты, оставляя свое мнение напоследок или не высказывать его совсем.

Системы управления проектами

К числу наиболее известных систем для управления проектами относятся:

- **MS Excel**. Хорошо подходит для недельного планирования и отчетности.
- **MS Project 2002**. Microsoft Project является на сегодня самой распространенной в мире системой управления проектами. Во многих западных компаниях MS Project стал привычной добавкой к Microsoft Office даже для рядовых сотрудников, которые используют его для планирования графиков несложных комплексов работ. Отличительной особенностью пакета является его простота. Разработчики MS Project не стремятся вложить в пакет сложные алгоритмы календарного или ресурсного планирования.

Семейство 2002 состоит из следующих продуктов:

- **MS Project Standard 2002** – рус. Легкая, универсальная система для управления проектами, в том числе для планирования и формирования графиков выполнения проектов. В сочетании с сервером MS Project Server 2002 позволяет наладить коллективную работу над проектом в масштабах рабочей группы на предприятиях разной величины.
- **MS Project Professional 2002**. Новое приложение. Содержит всю функциональность Microsoft Project Standard 2002 и в сочетании с Microsoft Project Server 2002 обеспечивает поддержку коллективной работы над проектами и предоставляет средства анализа и управления проектами и ресурсами в масштабах крупного предприятия.
- **MS Project Server 2002** – рус. Очередное пополнение в семействе Microsoft .NET Server, которое в сочетании с Microsoft Project Professional и Microsoft Project Standard обеспечивает полноценную поддержку коллективной работы над проектами, а также содержит средства анализа и управления ресурсами в масштабах всего предприятия.

- **MS Project Web Access 2002**. Web-интерфейс, предоставляющий доступ к информации о проектах и средствам анализа для руководителей и членов групп, которым не нужен полный набор функций управления в Microsoft Project. Эти пользователи могут обращаться к информации о проектах через Web-браузер. Поставляется в составе Microsoft Project Server 2002.

Open Plan. Производитель Welcom Corp. (США). Дистрибьютор в России ЛАНИТ. Open Plan – полностью руссифицированная система планирования и контроля крупных проектов и программ. Основные отличия системы:

- мощные средства ресурсного и стоимостного планирования,
- эффективная организация многопользовательской работы и
- возможность создания открытого, масштабируемого решения для всего предприятия.

Open Plan поставляется в двух вариантах – Professional и Desktop – каждый из которых отвечает различным потребностям исполнителей, менеджеров и других участников проекта.

- Продукты Primavera Systems, Inc. (США). Дистрибьютор в России ПМСОФТ:
  - **Primavera Project Planner**. Центральный программный продукт семейства Primavera Project Planner (P3) применяется для календарно-сетевого планирования и управления с учетом потребностей в материальных, трудовых и финансовых ресурсах средними и крупными проектами в самых различных областях, хотя наибольшее распространение данный продукт получил в сфере управления строительными и инженерными проектами.
  - **SureTrak Project Manager**. Кроме P3, компанией Primavera Systems поставляется облегченная система для УП - SureTrak. Этот полностью русифицированный продукт ориентирован на контроль выполнения небольших проектов или/и фрагментов крупных проектов. Может работать как самостоятельно, так и совместно с P3 в корпоративной системе управления проектами.
- **Spider Project**. Производитель Spider Technologies Group (Россия). Российская разработка Spider Project отличается мощными алгоритмами планирования использования ограниченных ресурсов и большим количеством дополнительных функций. Система спроектирована с учетом большого практического опыта, потребностей, особенностей и приоритетов Российского рынка. Spider Project поставляется в двух вариантах – Professional и Desktop.

- **Project Expert**. Производитель Про-Инвест Консалтинг (Россия). Российская разработка Project Expert обеспечивает построение финансовой модели предприятия, анализ финансовой эффективности бизнес-проектов, разработку стратегического плана развития и подготовку бизнес-плана.
- **1С-Рарус: Управление проектами**. 1С-Рарус (Россия). Российская разработка на платформе бухгалтерской системы "1С:Предприятие" версии 7.7 служит для планирования, организации, координации и контроля проектных работ и ресурсов. Типовое решение разработано только средствами и методами программы "1С: Предприятие" и представляет собой дополнение к компоненте "Бухгалтерский учет" программы "1С:Предприятие" версии 7.7. 1С-Рарус:Управление проектами интегрируется с любыми конфигурациями, которые используют компоненту 1С "Бухгалтерский учет".

КОНТРОЛЬНЫЕ ВОПРОСЫ ПО ДИСЦИПЛИНЕ «УПРАВЛЕНИЕ ПРОГРАММНЫМИ ПРОЕКТАМИ»

1.Проекты и управление проектами. Наука и искусство управления. Схема факторов и направлений, влияющих на УПП,

2.Операционная и проектная деятельность

3. Схема управления проектами (планирование, организация, мотивация, контроль)

4.Задачи оценки размера ПО. Когда они возникают? Размер ПО и трудоемкость ПО и факторы, влияющие на них. Связь между размером и трудоемкостью ПО.

5.Нелинейный характер связи между размером и трудоёмкостью ПО.

6.Метод функциональных точек. Влияние языка программирования.

7.Оценка размера ПО методом функциональных точек (ФТ). Чем оценивается сложность данных. Чем оценивается сложность транзакций.

8.«Выравнивающие факторы» для оценки размера ПО. Учёт этих факторов и уточнение числа ФТ.

9.Оценка трудоемкости программного проекта. Методика COCOMO11.Факторы масштаба проекта. Факторы среды разработки - множители трудоемкости

10.Оценка возможности реализации ПО в зависимости от размера в числе ФТ.

11.Процессы разработки ПО. Модели жизненного цикла ПО

12.Модель зрелости разработчика ПО (SWCMM). Пять уровней зрелости процессов разработки ПО (начальный, повторяемый, определенный, управляемый, оптимизируемый)

13.Модель PSP (Personal Software Process). Выбор модели процесса разработки при планировании.

14.Декомпозиция (разбиение) СТС на подсистемы – универсальный метод снижения сложности проектирования. Аутсорсинг.

15.Параллельная разработка подсистем и последовательная сборка системы.

16.Организация разработки ПО. Организация разработки в большом. Факторы успеха программного проекта. Причины неудач программных проектов

17.Норма управляемости. Причины выделения подразделений. Проект и организационная структура компании. Функциональная структура организации- разработчика ПО.

18. Проектная структура организация-разработчика ПО.

19. Матричная структура организации –разработчика ПО.

20.Планирование. Задачи планирования. Когда начинать планировать? Планирование от трудоемкости разработки.

21. Сетевые графики и их топология. Критический путь сетевого графика.

22.Сроки графика разработки ПО и вопросы их коррекции.

23..Диаграммы Ганта.

24.Риски - нештатные ситуации процесса разработки. *О*тчего возникают риски.

25.Планирование управления рисками. Идентификация рисков

26.Наиболее распространенные риски программных проектов

27. Характеристики процессов контроля, принципы контроля**.** Как проверять планы и как оценивать ход их исполнения?

28. Лидерство и формально правовая власть. Компетенции эффективного руководителя.

29.Стратегии руководства.

30.Управление персоналом, мотивация. Конфликт и управление проектом в этих условиях

31.Коммуникации при управлении программными проектами. Шкала эффективности коммуникаций.

32. Достижение компромисса и консенсуса**.** Как добиться консенсуса?

ПРИЛОЖЕНИЕ1

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРАКТИЧЕСКИМ ЗАНЯТИЯМ ПО ДИСЦИПЛИНЕ «УПРАВЛЕНИЕ ПРОГРАММНЫМИ ПРОЕКТАМИ»

Цель занятий – освоить метод функциональных точек (ФТ) и СОСОМО и оценить объем и трудоемкость разработки ПО, интерфейс и характеристики которого известны (курсовой проект, лабораторные работы и т.п.), сравнить полученный расчетный размер ПО с фактическим.

Метод ФТ использует для оценки объёма ПО его логическую модель, учитывающую количество и структуру входных и выходных данных, сложность функционала и интерфейса. Несомненным достоинством метода ФТ является то, что оценки количества ФТ не зависят от языка программирования и технологической платформы, на которой будет разрабатываться программный продукт. Учёт выразительности языка программирования проводится при переходе от оценок количества ФТ к оценкам числа строк исходного текста на ЯП.

В качестве оцениваемой программы в практических работах целесообразно брать разработанную программу с хорошо известным интерфейсом и функционалом, объёмом примерно 1000-2000 строк на языке программирования (ЯП), которая сделана самостоятельно в процессе выполнения лабораторных работ, курсовых проектов и т.п.

При планировании разработки ПО требуется провести оценку его трудоемкости, которая связывается с объёмом программы и характеристиками среды разработки. Эти оценки требуется провести в самом начале разработки до начала программирования задачи. При этом данные оценки производятся не на пустом месте: входные и выходные данные, примерный алгоритм, интерфейс с пользователем, характеристики среды разработки известны до начала программирования. В этой ситуации оценивается функциональный показатель объема ПО не в строках исходного текста на ЯП, а в количестве так называемых функциональных точек, как меры функциональности, **т.к. функциональность ПО не зависит от ЯП.**

Функциональные точки включают в себя подпрограммы выполнения небольших законченных функций в составе оцениваемой

задачи, а также некоторое её программное обрамление, предназначенное для связи данных функций с программным окружением, в котором эти функции существуют и пользователем. То есть оценки размера программы производятся с учетом ввода-вывода данных конкретного взаимодействия с пользователем, файлами баз данных и т.д.

Создатели метода ФТ проделали большую работу по определению возможностей перевода размера ПО, оцениваемого количеством ФТ в размер ПО, оцениваемый количеством строк исходного текста на языке программирования (ЯП). Опубликованы данные, определенные по статистике огромного количества ПО на разных языках программирования [10, 3]. Эти данные характеризуют эффективность ПО, а также дают ориентировку в среднем размере функциональной точки на разных ЯП.

Для оценки размера ПО, а затем его трудоемкости, конечно, можно использовать собственную статистику по данным характеристикам. Если же собственный опыт аналогичных проектов отсутствует, то не остается ничего другого, как использовать формальные методики, основанные на обобщенном опыте разработчиков ПО. Среди них наибольшее распространение получили два подхода:

- метод функциональных точек определения размера программы (метод ФТ),
-метод определения трудоемкости программного проекта COCOMO II (Constructive Cost Model).

Оценка размеров ПО методом функциональных точек (ФТ).

Для проведения оценок размера программы методом ФТ используются [10]:

1 .Базовый размер ФТ из таблицы 1.

2 Логическая схема оцениваемой программы и её интерфейс с пользователем и программным окружением.

3.Количество ФТ, определённых для оцениваемого ПО, учитывающее факторы сложности ПО.

. Метод разработан Аланом Альбрехтом (Alan Albrecht) в середине 70-х. Метод был впервые опубликован в 1979 году. В 1986

году была сформирована Международная Ассоциация Пользователей Функциональных Точек (International Function Point User Group — IFPUG), которая опубликовала несколько ревизий метода.

Таблица1. Размер ФТ в числе строк исходного текста на ЯП

Язык программирования	Минимальное число строк исходного текста на ЯП на одну ФТ	максимальрное количество строк исходного текста ЯП на одну ФТ
Ассемблер		330
C		128-150
Basic		105-107
Fortrun		105-107
Algol		105-107
Modula 2		80
Fortrun 95		71
Fort		64
Prolog		64
LISP		64
C++		64
Java		51
ADA95		49
Delphi		30
Visual Basic		32
Excel		6

Следует иметь ввиду различную **избыточность машинного кода**, для различных языков программирования и трансляторов с них. Эта избыточность связана с несовершенством компиляторов. В нашей таблице мы эту избыточность не учитывали – приведены строки исходного текста на языке программирования до компиляции.

При анализе методом функциональных точек надо выполнить следующую последовательность шагов (Рис. 1):

1. Определение типа оценки.
2. Определение области оценки и границ продукта.

3. Подсчет функциональных точек, связанных с данными.

4. Подсчет функциональных точек, связанных с транзакциями.

5. Определение суммарного количества не выровненных функциональных точек (UFP).

6. Определение значения фактора выравнивания (FAV).

7. Расчет количества выровненных функциональных точек (AFP).

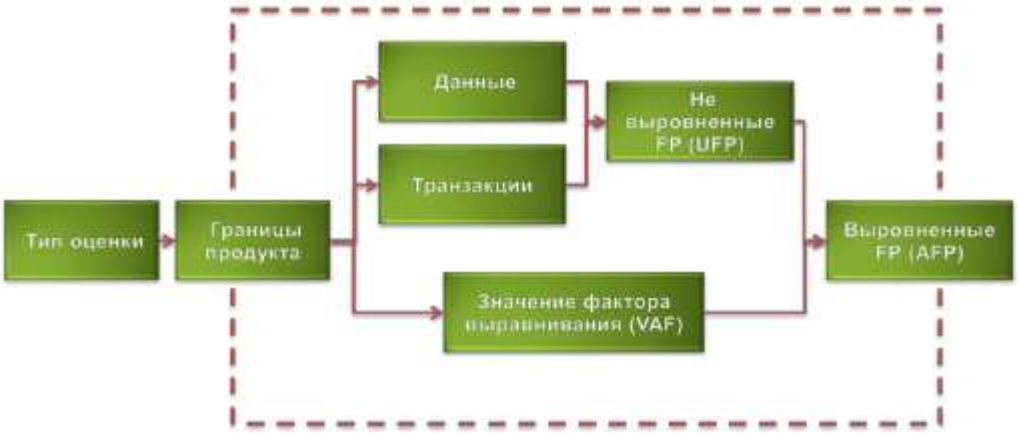


Рисунок 1. Процедура анализа по методу функциональных точек

Определение типа оценки

Первое, что необходимо сделать, это определить тип выполняемой оценки. Метод ФТ предусматривает оценки двух типов:

Проект разработки. Оценивается количество функциональности поставляемой пользователям в первом варианте продукта.

Проект развития. Оценивается в функциональных точках проект доработки: добавление, изменение и удаление функционала.

В практических занятиях используется проект разработки

Определение области оценки и границ продукта

Второй шаг — это определение области оценки. Область оценки включает все разрабатываемые функции (для проекта разработки или готового программного продукта) Оценка программного продукта методом ФТ вырабатывается не на пустом месте – рас-

сматриваются данные, которыми оперирует продукт и типы и количество его транзакций.- процессов преобразования данных (см. Рис2). Границы продукта определяются при составления логической или концептуальной модели продукта. При этом определяется:

-Что является «внешним» по отношению к оцениваемому продукту.

-Где располагается «граница системы», через которую проходят транзакции передаваемые или принимаемые продуктом, с точки зрения пользователя.

-Какие данные вырабатываются и поддерживаются в оцениваемой программе, а какие — внешние.

Рис. 2. Границы продукта в методе функциональных точек

К логическим данным системы относятся:

- Внутренние логические файлы (ILFs) — выделяемые пользователем логически связанные группы данных или блоки управляющей информации, которые поддерживаются внутри продукта.
- Внешние интерфейсные файлы (EIFs) — выделяемые пользователем логически связанные группы данных или блоки управляющей информации, на которые ссылается продукт, но которые поддерживаются вне продукта.

Примером логических данных (информационных объектов) могут служить: клиент, счет, тарифный план, услуга.

Подсчет функциональных точек, связанных с данными

Следующий шаг — подсчет функциональных точек, связанных с данными. Сначала определяется сложность данных по следующим показателям:

- **DET (data element type) — неповторяемое уникальное поле данных,** например, Имя Клиента — 1 DET; Адрес Клиента (индекс, страна, область, район, город, улица, дом, корпус, квартира) — 9 DET's
- **RET (record element type) — логическая группа данных,** например, адрес, паспорт, телефонный номер.

Оценка количества не выровненных функциональных точек, зависит от сложности данных, которая определяется на основании матрицы сложности (Таблица 2).

Таблица 2. Матрица сложности данных

	1-19 DET	20-50 DET	50+ DET
1 RET	Low	Low	Average
2-5 RET	Low	Average	High
6+ RET	Average	High	High

Оценка данных в не выровненных функциональных точках (UFP) подсчитывается по-разному для внутренних логических файлов (ILFs) и для внешних интерфейсных файлов (EIFs) (Таблица 3) в зависимости от их сложности.

Таблица 3. Оценка данных в не выровненных функциональных точках (UFP) для внутренних логических файлов (ILFs) и внешних интерфейсных файлов (EIFs)

Сложность данных	Количество UFP (ILF)	Количество UFP (EIF)
Low	7	5
Average	10	7
High	15	10

Для иллюстрации рассмотрим пример оценки в не выровненных функциональных точках объекта данных «Клиент» (Рис. 3).

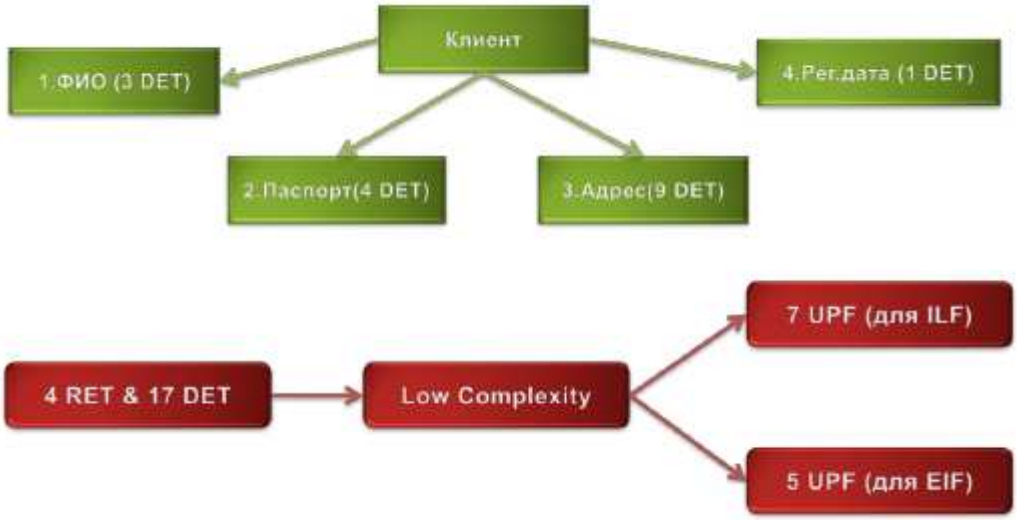


Рисунок 3. Пример оценки в не выровненных функциональных точках объекта данных «Клиент».

Объект «Клиент» содержит четыре логических группы данных, которые в совокупности состоят из 15 неповторяемых уникальное полей данных. Согласно матрице (Таблица 2), нам следует оценить сложность этого объекта данных, как «Low». Теперь, если оцениваемый объект относится к внутренним логическим файлам, то согласно Таблице 3 его сложность будет 7 не выровненных функциональных точек (UPF). Если же объект является внешним интерфейсным файлом, то его сложность составит 5 UPF.

Подсчет числа функциональных точек, связанных с транзакциями

Подсчет функциональных точек, связанных с транзакциями – это четвертый шаг анализа по методу функциональных точек.

Транзакция — это элементарный неделимый замкнутый процесс, представляющий значение для пользователя и переводящий продукт из одного состояния в другое.

В методе различаются следующие типы транзакций (Таблица 4):

- **EI (external inputs) —** внешние **входные** транзакции, элементарная операция по обработке данных или управляющей информации, поступающих в систему из вне.
- **EO (external outputs) —** внешние **выходные** транзакции, элементарная операция по генерации данных или управляющей информации, которые выходят за пределы системы.

Предполагает определенную логику обработки или вычислений информации из одного или более ILF.

- **EQ (external inquiries) — внешние** запросы, элементарная операция, которая в ответ на внешний запрос извлекает данные или управляющую информацию из ILF или EIF.

Таблица 4. Основные отличия между типами транзакций (О– основная; Д — дополнительная; NA — не применима).

Функция	**Тип транзакции**		
	EI	**EO**	**EQ**
Изменяет поведение системы	**О**	**Д**	**NA**
Поддержка одного или более ILF	**О**	**Д**	**NA**
Представление информации пользователю или Программному окружению	**Д**	**О**	**О**

Оценка сложности транзакции основывается на следующих ее характеристиках:

- FTR (file type referenced) — позволяет подсчитать количество различных файлов (информационных объектов) типа ILF и/или EIF модифицируемых или считываемых в транзакции.

- DET (data element type) — неповторяемое уникальное поле данных**.**

Примеры:

EI: поле ввода, кнопка, элементарная команда управления, имеющаяся в интерфейсе оцениваемого ПО с пользователем или программным окружением.

EO: поле данных отчета, операция по представлению результатов пользователю или программному окружению, сообщение об ошибке.

EQ: поле ввода для поиска, поле вывода результата поиска, операция, которая в ответ на внешний запрос выдаёт информацию

Для оценки сложности транзакций служат матрицы, которые представлены в Таблица 5 и Таблица 6.

Таблица 5. Матрица сложности внешних входных транзакций (EI)

EI	1-4 DET	5-15 DET	16+ DET
0-1 FTR	Low	Low	Average
2 FTR	Low	Average	High
3+ FTR	Average	High	High

Таблица 6. Матрица сложности внешних выходных транзакций и запросов (EO & EQ)

EO & EQ	1-5 DET	6-19 DET	20+ DET
0-1 FTR	Low	Low	Average
2-3 FTR	Low	Average	High
4+ FTR	Average	High	High

Оценка транзакций в не выровненных функциональных точках (UFP) может быть получена из матрицы (Таблица 7)

Таблица 7. Сложность транзакций в не выровненных функциональных точках (UFP)

Сложность транзакций	Количество UFP (EI & EQ)	Количество UFP (EO)
Low	3	4
Average	4	5
High	6	7

В качестве примера, рассмотрим оценку управляющей транзакции (EI) для диалогового окна, задающего параметры проверки орфографии в MS Office Outlook (Рис. 4).

Каждый "Check box" оценивается, как 1 DET. Выпадающий список — 1 DET. Каждая управляющая кнопка должна рассматриваться как отдельная транзакция. Например, если оценивать управляющую транзакцию по кнопке «OK», то, для данной транзакции мы имеем 1 FTR и 8 DET. Поэтому, согласно матрице (Таблица 5), мы можем оценить сложность транзакции, как Low. И, наконец, в соответствие с матрицей (Таблица 7), данная транзакция должна быть оценена в 3 не выровненных функциональных точек (UFP).

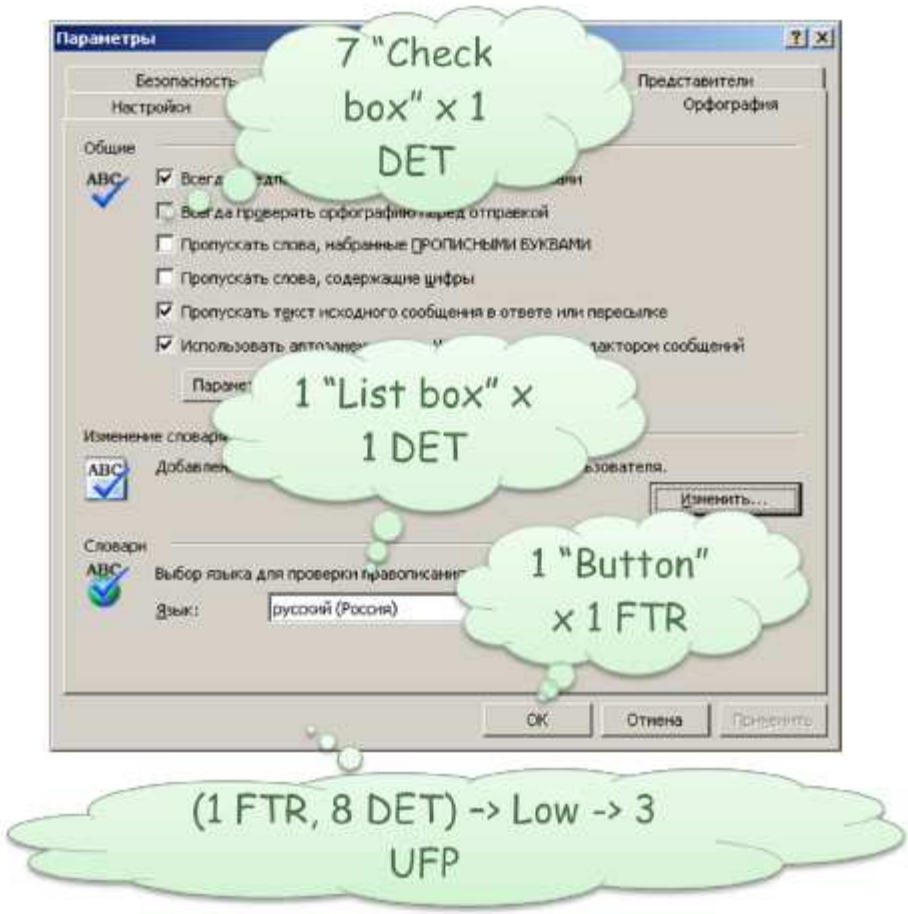


Рис. 4. Диалоговое окно, управляющее проверкой орфографии в MS Office Outlook

Определение суммарного количества не выровненных функциональных точек (UFP)

Общий объем продукта в не выровненных функциональных точках (UFP) определяется путем суммирования по всем информационным объектам (ILF, EIF) и элементарным операциям (транзакциям EI, EO, EQ).

Определение значения фактора выравнивания (VAF)

Помимо функциональных требований на продукт накладываются общесистемные требования, которые ограничивают разработчиков в выборе решения и увеличивают сложность разработки. Для учета этой сложности применяется фактор выравнивания (VAF). Значение фактора VAF зависит от 14 параметров, которые учитывают индивидуальные системные характеристики продукта:

1.*Обмен данными* (0 — продукт представляет собой автономное приложение; 5 — продукт обменивается данными по более, чем одному телекоммуникационному протоколу).

2.*Распределенная обработка данных (0* — продукт не перемещает данные; 5 — распределенная обработка данных выполняется несколькими компонентами системы).

3.*Производительность (0* — пользовательские требования по производительности не установлены; 5 — время отклика сильно ограничено критично для всех бизнес-операций, для удовлетворения требованиям необходимы специальные проектные решения и инструменты анализа.

4.*Ограничения по аппаратным ресурсам (0* — нет ограничений; 5 — продукт целиком должен функционировать на определенном процессоре и не может быть распределен).

5.*Транзакционная нагрузка (0* — транзакций не много, без пиков; 5 — число транзакций велико и неравномерно, требуются специальные решения и инструменты).

6.*Интенсивность взаимодействия с пользователем (0* — все транзакции обрабатываются в пакетном режиме; 5 — более 30% транзакций — интерактивные).

7.*Эргономика* (эффективность работы конечных пользователей) (0 — нет специальных требований; 5 — требования по эффективности очень жесткие).

8.*Интенсивность изменения данных* (ILF) пользователями (0 — не требуются; 5 — изменения интенсивные, жесткие требования по восстановлению).

9.*Сложность обработки (0* — обработка минимальна; 5 — требования безопасности, логическая и математическая сложность, многопоточность).

10.*Повторное использование* (0 — не требуется; 5 — продукт разрабатывается как стандартный многоразовый компонент).

11.*Удобство инсталляции (0* — нет требований; 5 — установка и обновление ПО производится автоматически).

12.*Удобство администрирования* (0 — не требуется; 5 — система автоматически самовосстанавливается).

13.*Мобильность (портируемость)* (0 — продукт имеет только 1 инсталляцию на единственном процессоре; 5 — система является распределенной и предполагает установку на различные «железо» и ОС).

14.*Гибкость (0* — не требуется; 5 — гибкая система запросов и построение произвольных отчетов, модель данных изменяется пользователем в интерактивном режиме).

Таким образом, 14 системных параметров (degree of influence - DI) оцениваются по шкале от 0 до 5. Расчет суммарного эффекта 14 системных характеристик (total degree of influence, TDI) осуществляется простым их суммированием:

$$TDI = \sum DI$$

Расчет значения фактора выравнивания производится по формуле

$$VAF = (TDI * 0.01) + 0.65$$

Например, если, каждый из 14 системных параметров получил оценку 3, то их суммарный эффект составит TDI = 3 * 14 = 42. В этом случае значение фактора выравнивания будет: VAF = (42 * 0.01) + 0.65 = 1.07. Суммарное влияние процедуры выравнивания лежит в пределах **±35% относительно объема, рассчитанного в UFP.**

Расчет количества выравненных функциональных точек (AFP) и расчёт размера ПО в строках текста на языке программирования

Оценка количества выровненных функциональных точек для программного приложения определяется по следующей формуле:

$$AFP = UFP * VAF.$$

Для получения размера ПО в числе строк исходного текста на ЯП необходимо умножить полученное количество выравненных ФТ *AFP* на размер одной ФТ на ЯП из таблицы1.

Метод анализа функциональных точек ничего не говорит о трудоемкости разработки оцененного продукта. Вопрос решается просто, если компания разработчик имеет собственную статистику трудозатрат на реализацию функциональных точек. Если такой статистики нет, то для оценки трудоемкости и сроков проекта можно использовать метод COCOMO II.

Основы методики COCOMO II

Методика COCOMO позволяет оценить трудоемкость и время разработки программного продукта. Впервые была опубликована

Бари Боэмом в 1981 году в виде результат анализа 63 проектов компании «TRW Aerospace». В 1997 методика была усовершенствована и получила название COCOMO II. Калибровка параметров производилась по 161 проекту разработки. В модели используется формула регрессии с параметрами, определяемыми на основе отраслевых данных и характеристик конкретного проекта [5,11].

Различаются две стадии оценки проекта: *предварительная* оценка на начальной фазе и *детальная* оценка после проработки архитектуры. Мы в соответствии с целями практического занятия рассматриваем предварительную оценку на начальной фазе.

Формула оценки трудоемкости проекта в чел.*мес. имеет вид:

$$PM = A \times SIZE^{E} \times \prod_{i=1}^{n} EM_i$$

$$A = 2{,}94$$

$$E = B + 0{,}01 \times \sum_{j=1}^{5} SF_j$$

$$B = 0{,}91$$

Здесь

SIZE — размер продукта в KSLOC

EM_i — множители трудоемкости

SF_j — факторы масштаба

n=7 — для предварительной оценки

n=17 — для детальной оценки

Главной особенностью методики является то, что для того, чтобы оценить трудоемкость, необходимо знать **размер программного продукта в тысячах строках исходного кода на ЯП (KSLOC, Kilo Source Lines Of Code).** Размер программного продукта может быть, например, оценен экспертами или же получен методом ФТ.

Факторы масштаба

В методике используются пять факторов масштаба SF_j, которые определяются следующими характеристиками проекта:

1.PREC — прецедентность, наличие опыт аналогичных разработок (Very Low — опыт в продукте и платформе отсутствует; Extra High — продукт и платформа полностью знакомы)

2.FLEX — гибкость процесса разработки (Very Low — процесс строго детерминирован; Extra High — определены только общие цели).

3.RESL — архитектура и разрешение рисков (Very Low — риски неизвестны/не проанализированы; Extra High — риски разрешены на 100%)

4.TEAM — сработанность команды (Very Low — формальные взаимодействия; Extra High — полное доверие, взаимозаменяемость и взаимопомощь).

5.PMAT — зрелость процессов (Very Low — CMM Level 1; Extra High — CMM Level 5) . **Для проектов учебного ПО(курсовые проекты, лабораторные работы) уровень принимать Very Low.**

Значение фактора масштаба, в зависимости от оценки его уровня, приведены в Таблице 8.

Таблица 8. Значение фактора масштаба, в зависимости от оценки его уровня

Фактор масштаба	Оценка уровня фактора					
	Very Low	Low	Nominal	High	Very High	Extra High
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

Множители трудоемкости

В нашей задаче рассматривается только случай предварительной оценки трудоемкости программного проекта. Для этой оценки необходимо оценить для проекта уровень семи множителей трудоемкости:

1.**PERS** — квалификация персонала (Extra Low — аналитики и программисты имеют низшую квалификацию, текучесть больше 45%; Extra High — аналитики и программисты имеют высшую квалификацию, текучесть меньше 4%)

2.**RCPX** — сложность и надежность продукта (Extra Low — продукт простой, специальных требований по надежности нет, БД маленькая, документация не требуется; Extra High — продукт очень сложный, требования по надежности жесткие, БД сверхбольшая, документация требуется в полном объеме). **Для проектов учебного ПО(курсовые проекты, лабораторные работы) уровень принимать Very Low.**

3.RUSE — разработка для повторного использования (Low — не требуется; Extra High — требуется повторное использование в других продуктах). **Для проектов учебного ПО (курсовые проекты, лабораторные работы) уровень принимать Low.**

4.PDIF — сложность платформы разработки (Extra Low — специальные ограничения по памяти и быстродействию отсутствуют, платформа стабильна; Extra High — жесткие ограничения по памяти и быстродействию, платформа нестабильна).

5.**PREX —** опыт персонала (Extra Low — новое приложение, инструменты и платформа; Extra High — приложение, инструменты и платформа хорошо известны).

6.**FCIL —** оборудование (Extra Low — инструменты простейшие, коммуникации затруднены; Extra High — интегрированные средства поддержки жизненного цикла, интерактивные мультимедиа коммуникации).

7.**SCED — сжатие** расписания (Very Low — 75% от номинальной длительности; Very High — 160% от номинальной длительности). **Для проектов учебного ПО(курсовые проекты, лабораторные работы) уровень принимать Nominal.**

Влияние множителей трудоемкости в зависимости от их уровня определяется их числовыми значениями, которые представлены в матрице, приведенной ниже, (Таблица 9).

Таблица 9. Значения множителей трудоемкости, в зависимости от оценки их уровня

	Оценка уровня множителя трудоемкости

	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
PERS	**2.12**	**1.62**	**1.26**	**1.00**	**0.83**	**0.63**	**0.5**
RCPX	**0.49**	**0.60**	**0.83**	**1.00**	**1.33**	**1.91**	**2.72**
RUSE	**n/a**	**n/a**	**0.95**	**1.00**	**1.07**	**1.15**	**1.24**
PDIF	**n/a**	**n/a**	**0.87**	**1.00**	**1.29**	**1.81**	**2.61**
PREX	**1.59**	**1.33**	**1.22**	**1.00**	**0.87**	**0.74**	**0.62**
FCIL	**1.43**	**1.30**	**1.10**	**1.0**	**0.87**	**0.73**	**0.62**
SCED	**n/a**	**1.43**	**1.14**	**1.00**	**1.00**	**1.00**	**n/a**

Из этой таблицы, в частности, следует, если в нашем проекте низкая квалификация аналитиков, то его трудоемкость возрастет примерно в 4 раза по сравнению с проектом, в котором участвуют аналитики экстра-класса. И это не выдумки теоретиков, а статистика!

По практической работе выпускается отчет, шаблон для которого приведен ниже

ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ ПРАКТИЧЕСКОЙ РАБОТЫ ПО УПП

ГРУППА

ФИО студента

Название программы

1.Привести логическую схему оцениваемой программы с указанием количества ILF,EIF, транзакций и их типов

2. Привести подсчет функциональных точек, связанных с данными.

указывается сложность данных по следующим показателям:

DET (data element type) — неповторяемое уникальное поле данных

RET (record element type) — логическая группа данных

3.Привести сложность данных на основании матрицы сложности (Таблица 2) .отдельно для ILF,EIF,

4.Привести оценку данных в не выровненных функциональных точках (UFP) раздельно для ILF,EIF (Таблица 3)

5. **Привести подсчет функциональных точек, связанных с транзакциями**

Привести FTR (file type referenced) – подсчитать количество различных файлов (информационных объектов) типа ILF и/или EIF модифицируемых или считываемых в транзакции.

6.Привести DET (data element type) — неповторяемое уникальное поле данных**.**

7.Указать сложность транзакций. Для оценки сложности транзакций служат матрицы, которые представлены в Таблице 5 и Таблице 6.

8. Привести оценку транзакций в не выровненных функциональных точках (UFP), полученную из матрицы (Таблица 7)

9. Привести общий объем продукта в не выровненных функциональных точках (UFP). Определяется путем суммирования определенных не выровненных ФТ по всем информационным объектам (ILF, EIF) и элементарным операциям (транзакциям EI, EO, EQ).

10. Привести значения фактора выравнивания (***VAF***)

Расчет значения фактора выравнивания производится по формуле

$$VAF = (TDI * 0.01) + 0.65 \qquad TDI = \sum DI$$

11.Привести количество выравненных функциональных точек (AFP)

*AFP = UFP * VAF.*

12.Привести размер ПО в числе строк исходного текста на ЯП. Для этоого необходимо умножить полученное количество выравненных ФТ (*AFP*)на размер одной ФТ на ЯП.

13 Привести фактическое значение числа строк программы на ЯП

14.По формуле оценить трудоемкость проекта в чел.*мес. **Привести PM, E,** П **EM_i**

$$PM = A \times SIZE^E \times \prod_{i=1}^{n} EM_i$$

$$A = 2{,}94$$

$$E = B + 0{,}01 \times \sum_{j=1}^{5} SF_j$$

$$B = 0{,}91$$

ПРИЛОЖЕНИЕ2

Аппроксимация полученных данных по фактическому и расчётному размеру программы методом наименьших квадратов и интерпретация результатов

Для каждой оцениваемой программы на плоскости X,Y можно поставить точку с координатами x_{i-}-фактический размер программы и y_i – размер программы, определённый по методу ФТ. Через «облако» точек надо провести аппроксимирующую прямую, так , чтобы она удовлетворяла в среднем всем точкам. Если эта прямая составляет с осью X угол ~45 град, значит в среднем метод ФТ работает хорошо.

Вообще подбор зависимостей, аппроксимирующих экспериментально полученные данные методом наименьших квадратов(МНК) преследует следующие цели:

1.Сглаживать данные, устраняя случайные и неизбежные ошибки измерений.

2.Получать значения данных лежащих между значениями аргументов их представления –интерполировать данные внутри интервала представления,

3. Экстраполировать данные за интервал измерения, предсказывая их поведение на будущие значения аргумента.

4.Устанавливать закономерности и зависимости между данными, имеющими случайную природу.

Эта работа успешно завершается, особенно относительно третьей задачи, если имеются теоретические предпосылки, обосновывающие вид аппроксимирующей кривой. Часто графическое представление экспериментальных данных позволяет принять «на глаз» вид аппроксимирующей кривой. Для простоты мы будем эту зависимость представлять линейной функцией.

Пусть имеются экспериментальные данные, сведенные в таблицу

y_i	x_i

Аппроксимирующая эти данные прямая описывается уравнением

$$y = b\,x$$

Сразу возникает проблема: у нас всего одно неизвестное b. Для его определения надо составить одно уравнение, использовав одно значение экспериментальных данных, а у нас их гораздо больше. Как определить какое-то среднее b, устраивающее всю совокупность исходных данных?

Дополним нашу таблицу **невязками Δ_i,** под которыми будем понимать разности между экспериментальными данными и данными с аппроксимирующей прямой на тех же значениях аргумента, что и у экспериментальных данных.

Невязки характеризуют точечные несовпадения экспериментальных y_i и данных с аппроксимирующей прямой y_{ai}, . Какова оценка несовпадения их по всей совокупности?

Сначала попробуем оценку проводить по сумме невязок.

$$\sum_5 \Delta_i$$

Здесь сразу видна проблема – невязки могут быть разных знаков и их сумма может быть маленькой даже нулевой, а сами невязки могут быть большими. Иными словами, сумма невязок плохо характеризует качество аппроксимации. Поэтому рассмотрим сумму модулей невязок либо сумму квадратов невязок. При этом в последнем случае большие невязки как бы подчеркиваются в сумме. В обоих случаях знак невязки не влияет на результат оценки при суммировании.

$$\sum_5 \backslash \Delta_i \backslash$$

$$\sum_5 \Delta_i^{\,2}$$

Как учитывать экспериментальные данные во всем диапазоне? Метод наименьших квадратов позволяет сделать это, определяя значения параметра b из условия **минимизации суммы квадратов невязок**. Составим сумму квадратов невязок

$$S=\sum_5 \Delta_i^{\,2}=\sum_5 (y_I - y_{ai})^2 = \sum_5 (y_I - bx_i)^2$$

Найдем минимум суммы квадратов невязок по значениям параметров b.

$$\frac{dS}{db} = 2\sum_{5} (y_I - bx_i)(-x_i) = 0$$

Раскрывая значения сумм имеем так называемое нормальное уравнение с одним неизвестным b. Значения сумм, как коэффициентов при неизвестном, легко вычисляются по исходным данным задачи. Никаких ограничений на число обрабатываемых данных нет.

$$\sum_{5} y_I x_i - b\sum_{5} x_i^2 = 0$$

Решение этого уравнения даёт выражение для b

$$b\sum_{5} x_i^2 = \sum_{5} y_I x_i$$

$$b = (\sum_{5} y_I x_i)) / (\sum_{5} x_i^2)$$

По условиям задачи суммы легко вычисляются

Если b близко к 1, то метод ФТ работает хорошо

Литература

1. Балашов А.И. Управление проектами : учебник и практикум/ А.И. Балашов, Е.М. Рогова, М.В. Тихонова, Е.А. Ткаченко; под общей редакцией Е.М. Рогвой. – М. : издательство Юрайт. 2015. – 383с.

2. В. А. Благодатских, В. А. Волнин, К. Ф. Поскакалов. Стандартизация разработки программных средств. Учебное пособие. Москва. Финансы и статистика. 2005г, 288 стр

3. Гецци К., Джазайери М., Мандриоли Д. Основы инженерии программного обеспечения.2-е изд.: Пер. с англ. – СПб: БХВ-Петербург, 2005. _ 832 с. 148 стр.

4. Д.И. Козлов, Г.П. Аншаков, Я.А. Мостовой, А.В. Соллогуб. Управление космическими аппаратами зондирования Земли. Компьютерные технологии. - М.: Машиностроение. 1998г.

5. Котов. С.Л. Нормирование жизненного цикла программной продукции. –М.: ЮНИТИ- ДАНА. 2002.–143с.

6. Мостовой Я.А. Лекции по технологии разработки программного обеспечения. Учебное пособие. Издательство ПГУТИ. Самара.2014г.178 с.

7.Макконел С. Совершенный код. Мастер класс/ Пер. с англ. – М. : Издательско-торговый дом» Русская редакция»; СПб.: Питер, 2005. – 896 стр.

8.Тьетар Р.-А. Менеджмент пер. с франц. – СПБ: Издательский дом «Нева», 2003, - 96с.

9.Терехов А.Н. Технология программирования: учеб. Пособие/А.Н. Терехов. 2-е изд., -М.: Интернет Университет информационных технологий; БИНОМ. Лаборатория знаний,2007.

10.Боэм Б.У. Инженерное проектирование программного обеспечения:Пер. с акнгл.– М.: Радио и связь. 1985. – 512с.

Интернет-ресурсы

11. Архипенков С. Лекции по управлению программными проектами. /www/arkhipenkov.ru.Москва. 2009

12. Орлик С. Инженерия ПО / http://sorlik.blogspot.com, http://SWEBOK.Sorlik.ru