Технология разработки программного обеспечения

Востриков Александр Владимирович

avostrikov@hse.ru

sanchs@inbox.ru

к.т.н., стар. преп. департамента компьютерной инженерии

Контрольные точки работы

- 1 модуль. Презентация должна содержать в себе постановку задачи (1 балл), обоснование актуальности проекта (1 балла), разработанное техническое задание (1 балл), бизнес-план проекта (2 балла), оценку текущего состояния проекта (1 балл). Дополнительные баллы проставляются за ответы на вопросы преподавателя (2 балла) и студентов (1 балл), качество выполнения презентации (1 балла).
- 2 модуль. Работающее ПО, презентация должна содержать в себе демонстрацию разработанного ПО (2 балла), расчет финансовых показателей проекта (1 балл), соответствие выполненных работ плану выполнения проекта (2 балла), оценку текущего состояния проекта и перспектив его развития (1 балл). Дополнительные баллы проставляются за ответы на вопросы преподавателя (2 балла) и студентов (1 балл), качество выполнения презентации (1 балла).

Рекомендуемая литература

- Орлов С.А. Технологии разработки программного обеспечения: Разработка сложных программных систем: Учебное пособие. – 3-е изд. – СПб.: Питер, 2004. – 526 с.
- Брукс Ф. Мифический человеко-месяц / Символ, С-Пб.: 2000.
- Липаев В.В. Системное проектирование сложных программных средств для информационных систем / Синтег, М.: 1999.
- Рейнвотер Дж. Как пасти котов. Наставление для программистов, руководящих другими программистами / СПб.: Питер. 2006. С. 256.
- Йордон Э. Путь камикадзе / Лори, М.: 2003.
- Глаголев В. Разработка технической документации. СПб.: Питер, 2008. – 192 с.
- ГОСТ 34.601-90
- ГОСТ Р ИСО/МЭК 12207 (ISO/IEC 12207).
- Благодатских В.А., Волнин В.А., Поскалоф К.Ф. Стандартизация разработки программных средств. М.: Финансы и статистика, 2007. – 288 с.

Лекция 1. Программное обеспечение компьютерных систем

1. ПО и его классификации

ПО – совокупность программ, выполняемых вычислительной системой. К ПО относится область деятельности по его проектированию и разработке:

- технология проектирования программ;
- методы тестирования программ;
- анализ качества работы программ;
- документирование программ.

Сфера применения ПО

- ПО современных компьютеров включает миллионы программ – от игровых до научных. ПО по назначению делится на:

- **Базовое** (системное) ПО;
- **Рабочее** (прикладное) ПО;
- **Инструментальное** ПО (средства разработки ПО – СУБД реляционные (Oracle, MySQL), объектно-ориентированные, иерархические, сетевые).

Классификация ПО по способу распространения

- **Коммерческое** ПО;
- **Бесплатные** программы;
- **Условно-бесплатные** (их можно получить и опробовать бесплатно, но для систематического пользования нужно платить);
- **Пиратские** (ворованные) копии программ (не имеют документации).

Пакеты прикладных программ

- **ППП** – комплект программ, предназначенных для решения задач в определенной области
- Выделяет следующие виды ППП:

- **проблемно-ориентированные** (где возможна типизация функций управления) – ППП автоматизации бухучета, управления персоналом;
- **Автоматизации проектирования** (в работе конструкторов – разработка чертежей);
- **Общего назначения** (графические редакторы, СУБД);
- **Офисные**;
- **Системы искусственного интеллекта** (экспертные системы, поддержка общения на естественном языке).

Программные средства и продукты

- **Программные средства** – математические средства, с помощью которых решаются задачи автоматизированного получения, обработки, хранения и выдачи информации.
- **Программный продукт** – совокупность отдельных программных средств, их документации, гарантий качества, рекламных материалов, мер по обеспечению пользователей, распространению и сопровождению готового ПО.
- **Программное изделие** – программа или логически связанная совокупность программ, записанная на носителях данных, являющаяся продуктом промышленного производства, снабженная программной документацией, предназначенная для широкого распространения посредством продажи.

При этом путь от «программ для себя» до программных продуктов достаточно долгий. Программные продукты могут создаваться как индивидуальная разработка под заказ,

- При **индивидуальной разработке** фирма-разработчик создает оригинальный программный продукт, учитывающий специфику обработки данных для конкретного заказчика.
- При разработке **для массового распространения** фирма-разработчик, с одной стороны, должна обеспечить универсальность выполняемых функций обработки данных, с другой стороны, гибкость и настраиваемость программного продукта на условия конкретного применения.

Программный продукт создается на основе промышленной технологии выполнения проектных работ с применением современных инструментальных средств программирования. Специфика заключается в уникальности процесса разработки алгоритмов и программ. На создание программных продуктов затрачиваются значительные ресурсы – трудовые, материальные, финансовые, требуется высокая квалификация разработчиков.

Как правило, программные продукты требуют сопровождения, которое осуществляется специализированными фирмами – распространителями программ (дистрибьюторами). **Сопровождение программ** массового применения сопряжено с большими трудозатратами – обнаружение и исправление ошибок

Рынок программных продуктов

На рынке ПП действуют:

- Поставщики ПП;
- Потребители ПП;
- Посредники.

В условиях существования рынка ПП **важными характеристиками** являются:

- Стоимость;
- Количество продаж;
- Время нахождения на рынке;
- Известность фирмы-разработчика и программы;
- Наличие ПП аналогичного назначения.

Маркетинг

ПП массового распространения продаются по ценам, которые учитывают конъюнктуру рынка (наличие и цены программ-конкурентов). Большое значение имеет проводимый фирмой маркетинг, который включает:

- Формирование политики цен для завоевания рынка;
- Широкую рекламную компанию ПП;
- Создание торговой сети для реализации ПП (дилеры и дистрибьюторы);
- Обеспечение сопровождения и гарантийного обслуживания ПП, создание горячей линии;
- Обучение пользователей ПП.

Спецификой ПП является также и то, что их эксплуатация должна выполняться на правовой основе – **лицензионные соглашения** между разработчиком и пользователями с соблюдением авторских прав разработчиков ПП.

Приобретение программного продукта – это покупка лицензии – права на его использование. Условия использования любого программного продукта описаны в лицензионном соглашении, которое представляет собой договор между производителем ПП и пользователем ПО. Для разных пользователей (индивидуальных покупателей, организаций разного масштаба, учебных и правительственных учреждений) могут быть установлены различные условия приобретения ПО.

Каждый пользователь ПП должен иметь лицензию на него. Лицензия должна быть закуплена для каждого компьютера, на котором установлен или используется через сеть ПП. Договор между пользователем и производителем не подписывается: считается, что покупатель соглашается с условиями лицензионного соглашения, если он вскрывает **дистрибутив** – упаковку с компакт-диском. Это так называемая «оберточная лицензия», предусмотренная Законом «О правовой охране

Лицензия

ПО на компьютере **находится в «пользовании»,** когда оно помещено в постоянную память (жесткий диск или флэшка, компакт диск) или загружено в оперативную память. В компьютерной сети ПП может использоваться одним из двух способов: запуск ПО с локального жесткого диска рабочей станции или установка ПП только на сервер сети и запуск ПО с сервера. Вне зависимости от того, как используется продукт в сети (с сервера или с локального рабочего места), каждый пользователь должен обладать лицензией на право использования этого продукта. Только такой вариант использования ПП является законным.

Приобретение лицензии

Существует несколько вариантов приобретения лицензии. Наиболее известный и распространенный путь – покупка коробки с ПП. Коробка содержит лицензионное соглашение, регистрационную карточку, дистрибутив с ПП и документацию. Если появляется необходимость в использовании этого ПП на других компьютерах, недостаточно приобрести одну коробку. В этом случае многие поставщики ПО предлагают приобрести только лицензию – конверт, содержащий лицензионное соглашение, цена которого ниже, чем цена коробки.

Лекция 2
Разработка ПС

Стадии разработки ПО, регламентированные ГОСТ

В РФ ЖЦ разработки ПО установлен стандартом ГОСТ 19.106-78 «Общие требования к программным документам, выполненным печатным способом» (09.1981) и содержит следующие стадии и этапы:

1. **ТЗ**
2. **Эскизный проект (ЭП)**
3. **Технический проект (ТП)**
4. **Рабочий проект (РП)**
5. **Внедрение**

Техническое задание

На стадии ТЗ выполняются следующие работы, входящие в состав соответствующих этапов.

1. **Обоснование необходимости разработки программ**: постановка задачи, сбор исходных материалов, выбор и обоснование критериев эффективности и качества.
2. **Выполнение НИР**: определение структуры входных и выходных данных, предварительный выбор методов решения задач, обоснование целесообразности применения ранее разработанных программ, определение требований к техническим средствам, обоснование возможности решения поставленных задач.
3. **Разработка и утверждение ТЗ**: определение требований к ПО, разработка технико-экономических показателей, определение стадий, этапов и сроков разработки ПО и документации на него, выбор языков программирования, согласование и утверждение ТЗ.

Эскизный проект

Результатом выполнения данной стадии является полное описание архитектуры ПО. Как правило, это описание делается на нескольких уровнях иерархии. На верхнем уровне детализации выделяются основные подсистемы, устанавливаются связи между основными подсистемами, прописываются функции подсистем. Затем процедура декомпозиции выполняется для каждой подсистемы, выделяются модули, составляющие эту подсистему. В итоге получается иерархически организованная система, состоящая из уровней (связь модулей). Единицы, выделяемые на различных уровнях, определяются разработчиком. Результаты ЭП отображаются в документе Пояснительная записка к ЭП, оформленному по ГОСТ 19.404-79.

Технический проект

Содержанием работ по этой стадии является проектирование структуры ПО. Результатом – реализующий заданный и утвержденный в ТЗ комплекс программ. Форма представления результата – пояснительная записка к техническому проекту согласно ГОСТ 19.105-78. Разработка структуры ПО заключается в выделении всех программных компонентов по функциональным признакам, определение функциональных спецификаций модулей, структуры входных и выходных данных, определение операционной среды, аппаратных средств.

Рабочий проект

Содержанием работ на этой стадии является описание ПО на выбранном проблемно-ориентированном языке (кодирование), разработка, отладка, согласование и утверждение порядка и методики испытаний, разработка программных документов, проведение тестирования, проведение приемо-сдаточных испытаний. Результат – ПО в форме программной документации или в форме программного изделия.

Качество ПО

Качество ПО – способность ПО подтвердить свою спецификацию при условии, что спецификация ориентирована на характеристики, которые желает получить пользователь.

Одной из важнейших проблем обеспечения качества программных средств является формализация характеристик качества и методология их оценки. Методологии и стандартизации оценки характеристик качества программных средств на различных этапах жизненного цикла посвящен международный стандарт ISO 14598.

Рекомендуется следующая общая схема процессов оценки характеристик качества программ:

Функциональная пригодность – наиболее неопределенная характеристика программного средства.

Оценка корректности программных средств состоит в формальном определении степени соответствия комплекса реализованных программ исходным требованиям контракта, ТЗ и спецификаций на программное средство. Путем верификации должно быть определено соответствие исходным требованиям всей совокупности компонентов комплекса программ, вплоть до модулей и текстов программ с описанием данных.

Оценка способности к взаимодействию состоит в определении качества совместной работы компонентов программных средств и БД с другими прикладными системами и компонентами на различных вычислительных платформах, а также взаимодействия с пользователями в стиле, удобном для перехода от одной вычислительной системы к

Оценка защищенности программных средств включает определение полноты использования доступных методов и средств защиты программного средства от потенциальных угроз и достигнутой при этом безопасности функционирования информационной системы. Наиболее широко и детально задачи оценки комплексной защиты информационных систем изложены в ISO 15408:1999-1-3 «Методы и средства обеспечения безопасности. Критерии оценки безопасности информационных технологий».

Оценка надежности – измерение количественных метрик к таким показателям как завершенность, устойчивость к дефектам, восстанавливаемость.

Потребность в ресурсах памяти и производительности компьютера в процессе решения задач значительно изменяется в зависимости от состава и объема исходных данных. Для корректного определения предельной пропускной способности информационной системы нужно измерить экстремальные и средние значения длительности исполнения программ.

Оценка практичности программных средств проводится экспертами и включает определение понятности, простоты использования, изучаемости и привлекательности программного средства.

Сопровождаемость можно оценивать полнотой и достоверностью документации на ПО. Она должна определять стратегию, стандарты, процедуры, распределение ресурсов и планы создания, изменения и применения документов на программы и данные.

Оценка мобильности – подготовленность ПС к переносу на другую платформу.

Надежность ПО

- **Надежность ПО** – способность ПП безотказно выполнять определенные функции при заданных условиях с большой вероятностью.
- **Степень надежности** характеризуется вероятностью работы ПП без отказа в течение определенного периода времени.

Источниками ненадежности являются непроверенные сочетания исходных данных, при которых отлаженные программы дают неверные результаты и отказы.

Отказ – нарушение работоспособности ПО, являющееся следствием таких явлений, как нарушение кодов записи программ в памяти, стирания или искажения данных в оперативной или долговременной памяти, нарушения нормального хода вычислительного процесса.

Сбой – самоустраняющийся отказ, не требующий внешнего вмешательства. Основное различие между сбоем и отказом – по временному показателю длительности восстановления. Если длительность восстановления больше какого-то порогового значения, то аномалию в работе ПО относят к отказам, иначе – к сбоям.

Правильность ПО

Понятие корректной программы рассматривается статически вне времени функционирования. Неправильность программы определяется вероятностью совмещения следующих событий:

- Попадания исходных данных в область непроверенных при отладке и испытаниях;
- Проявления ошибки в программе при обработке таких данных.

Правильность программы не зависит от динамики функционирования в реальном времени.

Надежная программа должна обеспечивать низкую вероятность отказа в процессе функционирования. **Количество ошибок в программе не имеет никакого отношения к ее надежности:**

- Число ошибок в программе – величина «ненаблюдаемая», наблюдаются не сами ошибки, а результат их появления.
- Неверное срабатывание программы может быть следствием не одной, а сразу нескольких ошибок.
- Ошибки могут компенсировать друг друга, так что после исправления какой-то одной ошибки программа может начать «работать хуже».
- Надежность характеризует частоту проявления ошибок, но не их количество. **Неверное**

Лекция 3

Жизненный цикл ПО

Жизненный цикл (ЖЦ) программной системы – это последовательность этапов, через которую проходит она в ходе своего существования. На данный момент существует несколько моделей жизненного цикла программных систем. В общем случае ЖЦ состоит из следующих этапов:

- анализ;
- проектирование;
- разработка;
- тестирование;
- развертывание;
- использование.

Варианты жизненного цикла программ

Варианты жизненного цикла программ

Также различают различные виды жизненных циклов и проектирования по виду сборки готового продукта. Это проектирование **нисходящее** (сверху вниз), **восходящее** (снизу вверх), **расширения ядра** (проектирование отдельных основных модулей, иерархическое проектирование дополнительных модулей, наращивающих функциональность) и **смешанное**.

При **проектировании сверху вниз** проводится общий анализ системы: определяются входные и выходные данные, требования к ним. Далее производится декомпозиция системы на отдельные подсистемы, определяются функциональность подсистем, связи между ними, наборы данных проходящих по этим связям, требования к данным. Далее процесс проектирования повторяется к отдельным подсистемам.

При **восходящем проектировании** на начальном этапе проектируются самостоятельные системы. После этого проектировщик пытается объединить их между собой таким образом, чтобы получить связанные подсистемы, обладающие большей функциональностью. Подобное объединение производится до тех пор, пока не будет получен модуль, обладающий требуемой функциональностью.

Метод расширения ядра заключается в том, что проектировщик выделяет наиболее важный, с его точки зрения, модуль и производит его проектирование. Далее проектируются модули, которые расширяют функциональность имеющегося в заданном направлении, производится их объединение. Таким образом, на каждом этапе проектирования в наличии имеется система с ограниченной функциональностью.

При **смешанных методиках** возможны комбинации указанных подходов. Так, например, в начале работ могут быть определены основные универсальные блоки, которые потребуются при создании системы. После их проектирования начинается проектирование сверху вниз, причем при проектировании функциональности блоков учитывается возможность включения в них уже разработанных универсальных.

Распределение работ

Существует эмпирический закон, который гласит, что в процессе создания ПО 30% времени тратится на анализ требований и проектирование, 40% времени – на кодирование и еще 30% - на доводку и тестирование. В конкретных случаях это отношение может меняться, однако для общего случая вполне верно. Можно даже сказать, что уровень программистов и разработчиков можно определить по тому, насколько соблюдается данное соотношение. При равномерно сбалансированной команде соотношение будет соблюдаться.

Анализ

Этап анализа требований посвящен работе с заказчиком. На данном этапе заказчик предъявляет требования к создаваемой системе по ее функциональности, качеству, времени и стоимости разработки. Требования к системе должны быть четко оговорены, задокументированы и подписаны обеими сторонам. Это делается для того, чтобы ни одна из сторон не могла в последствии отказаться от набора и характеристик требований: разработчик не может сдать не полный комплект работы, а заказчик не может претендовать на выполнение дополнительных работ или коррекцию их качества. Четкое формулирование требований избавляет как заказчика, так и разработчика от большого количества проблем, которые могут возникнуть впоследствии. Слишком общая постановка задачи приводит к множественности ее трактовки, слишком узкая постановка задачи лишает разработчиков маневра.

Двумя крайними случаями при анализе требований могут быть либо постановка задачи «Сделайте нам работающую систему», либо навязывание разработчикам приемов и методов, а также строгого собственного видения функциональности системы. В первом случае реакцией заказчика на готовый продукт может быть: «Это совсем не то, что мы ожидали. Мы думали, что вы профессионалы и можете разработать нормальную систему». Во втором случае может выясниться, что система не может быть разработана при существующих требованиях по соображениям технического (например, завышенные требования к производительности) либо экономического (например, быстрое создание сложной высококачественной хорошо задокументированной и передаваемой системы за минимальные деньги) плана.

Анализ. Разработка требований и внешнее проектирование ПО

1. Общая схема создания ПО

Процесс создания программ можно представить как последовательность действий:

1. Постановка задачи
2. Алгоритмизация решения задачи
3. Программирование

Постановка задачи

- это точная формулировка решения задачи на компьютере с описанием входной и выходной информации.

К основным характеристикам функциональных задач, уточняемым в процессе ее формализованной постановки, относятся:

- Цель и назначение задачи, ее место и связи с другими задачами;
- Условия решения задачи с использованием средств вычислительной техники;
- Содержание функций обработки входной информации при решении задачи;
- Требования к периодичности решения задачи;
- Ограничения по срокам и точности выходной информации;
- Состав и форма представления выходной информации;
- Источники входной информации для решения задачи;
- Пользователи задачи

Алгоритм

- Система точно сформулированных правил, определяющая процесс преобразования допустимых исходных данных в желаемый результат за конечное число шагов.

Обязательные свойства алгоритмов:

- Дискретность
- Определенность
- Выполнимость
- Массовость

В алгоритме обязательно должны быть предусмотрены все ситуации, которые могут возникнуть в процессе решения.

Программирование

Программа – реализованной алгоритм на языке программирования.

Наиболее часты программисты делятся на **системных и прикладных**.

В условиях создания больших по масштабам и функциям обработки программ существует квалификация – **программист-аналитик**, который анализирует и проектирует комплекс взаимосвязанных программ для реализации функций предметной области.

В процессе создания программ на начальной стадии работ участвуют и специалисты – **постановщики задач**.

Большинство информационных систем основано на работами с БД. Если БД является интегрированной, обеспечивающей работу с данными многих приложений, возникает проблема организационной поддержки БД, которая выполняется **администратором БД**.

Основным потребителем программ служит **конечный**

2. Разработка требований к ПО

Наиболее оптимальной является совместная работа проектировщиков и пользователей по выработке требований.

Можно установить 2 фазы по выработке требований:

1) **Фаза планирования**. Определяется реализуемость, устанавливаются цели, оцениваются затраты, обеспечивается ориентация для разработки проекта.
2) **Фаза выработки требований пользователя.** Вырабатываются требования к входным данным, информационным потокам, выходным данным, документации, среде, вычислительным ресурсам.

3. Цели разработки ПО

Цели разработки обычно включают следующую информацию:

- Краткое описание ПО
- Определение круга пользователей
- Подробное описание функциональных задач
- Документация. Определяются типы документации и предполагаемый круг читателей для каждого типа.
- Эффективность. Временные характеристики, использование ресурсов.
- Совместимость
- Конфигурация
- Безопасность
- Обслуживание
- Установка
- Надежность. Сбои и их последствия

4. Разработка внешних спецификаций проекта

Внешнее проектирование – это процесс описания планируемого поведения разрабатываемого ПО с точки зрения потенциальных пользователей (вопрос устройства интерфейса ПО).

Разработка внешних спецификаций разбивается на 2 части:

- Предварительный внешний проект
- Детальный внешний проект

Предварительный внешний проект содержит описание основных компонентов и внешних функций, составляющих отдельные компоненты проекта. Неопределенным остается точный синтаксис, семантика, выходные результаты.

Детальный внешний проект должен включать следующую информацию:

1. Описание входных данных (точное описание синтаксиса и семантики всех данных, вводимых пользователем).
2. Описание выходных данных (точное описание результатов функций).
3. Характеристики надежности (описание влияния всевозможных отказов).
4. Эффективность
5. Замечания по программированию

Лекция 5

Проектирование и разработка

Пользовательский интерфейс является своеобразным коммуникационным каналом, по которому осуществляется взаимодействие пользователя и компьютера.

Лучший пользовательский интерфейс – это такой интерфейс, которому пользователь не должен уделять много внимания или почти не замечать его.

Общие принципы проектирования пользовательских интерфейсов

- Программа должна помогать выполнить задачу, а не становиться этой задачей.
- При работе с программой пользователь не должен ощущать себя дураком.
- Программа должна работать так, чтобы пользователь не считал компьютер дураком.

Графический интерфейс пользователя

Графический интерфейс пользователя (GUI) является обязательным компонентом большинства современных программных продуктов, ориентированных на работу конечного пользователя. К графическому интерфейсу предъявляются высокие требования как с чисто инженерной, так и с художественной стороны разработки, при его разработке ориентируются на возможности человека.

Наиболее GUI реализуется в интерактивном режиме работы пользователя для программных продуктов, функционирующих в среде Windows, и строится в виде системы спускающихся меню с использованием в качестве средства манипуляции мыши и клавиатуры. Работа пользователя осуществляется с экранными формами, содержащими объекты управления, панели инструментов с пиктограммами режимов и команд обработки.

Формы

Формы – это строительные блоки интерфейса пользователя.

Формы делятся на два вида: формы ввода и формы справочники. В формах ввода создаются и редактируются документы. Форма ввода состоит из одного или нескольких блоков, как правило, в виде таблиц. В справочниках вводится справочная информация.

Виды интерфейса:

- Однодокументный (SDI);
- Мнодокументный (MDI).

В SDI-приложениях окна форм появляются совершенно независимо друг от друга.

Диалоговый режим

Большинство программных продуктов, особенно прикладного характера, ориентированных на конечного пользователя, работают в **диалоговом режиме** взаимодействия с пользователем таким образом, что ведется обмен сообщениями, влияющими на обработку данных.

В диалоговом режиме под воздействием пользователя осуществляется запуск функций (методов) обработки, изменение свойств объектов, производится настройка параметров выдачи информации на печать и т.п.

Системы, поддерживающие диалоговые процессы, классифицируются на:

- Системы с жестким сценарием диалога – стандартизированное представление информации обмена;
- Дескрипторные системы – формат ключевых слов сообщений;
- Тезаурусные системы (аналог – гипертекстовые системы);
- Системы с языком деловой прозы – представление сообщений на языке, естественном для профессионального пользования.

Диалоговые системы с жестким сценарием диалога

Наиболее просты для реализации и распространены диалоговые системы с жестким сценарием диалога, которые предоставлены в виде:

- Меню – диалог инициируется программой; пользователю предлагается выбрать вариант из перечня.
- Действия запрос-ответ – фиксирован перечень возможных значений, выбираемых из списка, или ответы типа Да/Нет;
- Запрос по формату – с помощью ключевых слов, фраз осуществляется подготовка сообщений.

Диалоговый процесс управляется согласно созданному сценарию, для которого определяются:

- Точки начала диалога
- инициатор диалога (человек или программа)
- параметры и содержание диалога

Разработка

Собственно **разработка ПО** заключается в детальном проектировании отдельных работ и их реализации. Обычно считается, что данный этап является основным с точки зрения реализации проекта в целом. Однако скорее можно сказать, что этап важен с точки зрения качества и функциональности продукта.

С точки зрения менеджера проекта при разработке программного обеспечения следует помнить, что кризисную ситуацию чаще проще предотвратить, чем найти пути выхода из нее. Это означает, что в ходе разработки следует внимательно относиться к выполнению всех сроков, требований по качеству и функциональности продукта, объему его функций. Менеджер проекта интересуется деталями хода проекта до наступления контрольных точек, чтобы к их наступлению все работы были завершены.

Также менеджеру следует помнить, что увеличение длительности рабочей недели не всегда положительно сказывается на производительности. Йордон приводит интересную статистику. При увеличении длительности рабочей недели до 60 часов производительность продолжает расти, от 60 до 100 часов в неделю производительность стабилизируется и начинает падать, при рабочей неделе более 120 часов производительность становится отрицательной, то есть программисты больше исправляют собственные ошибки, чем пишут новый код.

С точки зрения разработчика следует помнить, что качество программного кода закладывается именно в момент его написания. Чем меньше ошибок допустит сам программист, чем больше времени он потратит на проверку кода и его проработку, тем меньше шансов найти в нем какой-либо изъян. Зачастую здесь может помочь «метод пристального взгляда», когда уже по окончании разработки кода программист повторно просматривает собственный код, причем возможно даже в распечатках.

Выражаясь метафорично, основным девизом этапа разработки является «Контроль над всем». Менеджер должен контролировать соответствие разработки календарному плану и разработанным требованиям. Программисты должны следить за качеством и безопасностью разрабатываемого ими кода. Тестировщики должны отслеживать соответствие продукта его функциональности и требованиям. Составители документации и инженерные психологи должны контролировать удобство использования системы.

Тестирование ПО

Роль этапа **тестирования** зачастую незаслуженно принижается. Однако ошибки в программном коде – явление не только обычное, но и системное. В конце 80-х – начале 90-х был проведен ряд исследований в области эффективности труда программистов. Так, было выявлено, что программист со стажем более 10 лет совершает в среднем 131,3 ошибки на 1000 строк кода. 50% из них выявляется на этапе компиляции. На этапе тестирования выявляется половина из оставшихся ошибок. Для устранения оставшихся ошибок требуется от 10 до 40 человеко-часов на заключительных этапах. При этом устранение ошибки в готовом продукте обходится иностранным компаниям примерно в 4000$. Коэффициент обнаружения ошибок пользователями в готовом продукте к количеству ошибок, обнаруженных при тестировании составляет в среднем 0,91. То есть, если на тестирование поступает некачественный продукт, есть много шансов за то, что он таким и останется.

Для менее опытных программистов принята следующая статистика. Один программист делает в ходе работы от 0,5 до 3 ошибок на строчку кода. В результате устранения указанных в ходе работы компилятора ошибок в коде остается порядка 1 ошибки на 10 строчек кода. Отладка программы сокращает количество ошибок до 1 на 100 строк кода. Тестирование кода специально выделенным сотрудниками с последующей коррекцией кода снижает число ошибок до 1 на 1000 строк. Эти цифры принято брать за основу при планировании разработки.

По данным Microsoft устранение одной ошибки, требующей создания пресс-релиза и пакета обновлений, обходится фирме примерно в 100 000$.

Виды тестирования

Существует несколько видов тестирования. Начальное тестирование проводится непосредственно разработчиками для того, чтобы убедиться, что ПО работает в соответствии с документацией. Функциональное тестирование проводится с целью более глубокой проверки соответствия функциональности ПО. Нагрузочное тестирование проверяет работоспособность ПО при повышенных нагрузках. Тестирование пользовательского интерфейса может быть разделено на две части: проверка функциональности интерфейса и проверка удобства интерфейса. Наиболее общим является регрессионное тестирование, состоящее из всех тестов, которые программа проходила до сих пор. Подобное тестирование необходимо так как изменение функционирования некоторого модуля может повлиять на работу других, связанных с ним, модулей, зачастую – весьма многочисленных.

Под регрессионным тестированием понимается последовательный прогон всех тестов, которые запускались для данного продукта до сих пор. Задачей регрессионного тестирования является проверка функциональности уже включенных модулей. Модули, которые не прошли тестирование, отправляются на доработку.

Кроме того, тестирование делят на тестирование методом черного, серого и белого ящиков. В первом случае тестировщик не имеет никакой информации о структуре тестируемой системы и описывает только симптомы неисправности, при этом он может высказывать предположения о причинах возникновения неисправности. В последнем случае тестировщик обладает полным исходным кодом программы и может указывать конкретные строки кода, вызвавшие сбой. Тестирование методом серого ящика является промежуточным вариантом – тестировщик имеет представление о составе модулей тестируемой системы, однако не имеет информации об их внутренней структуре или особенностях реализации.

Функциональное тестирование преследует своей целью проверку корректности работы приложения. В таком виде тестирования основными задачами является проверка устойчивости, корректности и, возможно, безопасности системы.

Нагрузочное тестирование служит для проверки функционирования системы в экстремальных условиях: нехватка оперативной или внешней памяти, потери канала связи или соединения с другим приложением, работа с поврежденными файлами и так далее. При этом программа должна проявлять устойчивость и безопасность.

Тестирование функциональности пользовательского интерфейса проводится с целью определения соответствия действий, привязанных к элементам управления, требованиям, заявленным в документации. То есть любая функция должна быть доступна именно тем методом, который был заявлен.

Тестирование удобства пользовательского интерфейса проводится скорее инженерными психологами, чем тестировщиками. Здесь психологи проверяют насколько приятно и удобно пользоваться данной программой, очевиден ли ее интерфейс, нельзя ли его улучшить.

При исправлении ошибки следует помнить несколько стандартных подходов.

- Повторный просмотр кода резко снижает количество ошибок, обнаруженных на этапе тестирования.
- Обычно программисты делают примерно одни и те же «любимые» ошибки. При обнаружении одной из них проверьте код на предмет обнаружения других аналогичных. Здесь, как и везде, действует принцип 80/20 – 80% ошибок встречаются в 20% кода.
- Копирование кода является методом повышения производительности и количества ошибок.
- Нет ничего постояннее, чем временное. Если вы не запишите себе в список задач доработку временного кода, есть большая вероятность получить от отдела тестирования запись в свой список ошибок.

Развертывание ПО

Развертывание приобретает высокую актуальность для больших систем. В простейшем случае программа сдается заказчику на некотором носителе и не требует специальных действий для работы с ней. Однако для сложных систем для запуска комплекса может потребоваться целый ряд работ. Некоторые информационные системы, например, могут потребовать разворачивания высокопроизводительного кластера. Сама инсталляция аппаратного и программного обеспечения для его работы может быть весьма трудоемкой задачей. Далее может потребоваться установка и настройка специального программного обеспечения: серверов приложений и БД, клиентских программ, заполнение баз начальными данными. При развертывании принципиально нового программного обеспечения может потребоваться обучение персонала навыкам работы с ним. В результате необходимо будет провести комплекс учебных мероприятий для начала функционирования системы.

Например, сложные системы АСУ ТП обычно доводятся непосредственно у заказчика, так как различные предприятия обладают собственной спецификой. При внедрении нового оборудования и программных систем перед началом работ производится обучение персонала. В конечном итоге должна быть произведена приемка продукта с подписанием соответствующих актов.

Развертывание сложной системы может оказаться трудоемким и затратным процессом, который необходимо предусмотреть еще на этапах анализа требований и проектирования. Так, например, может оказаться, что заказчиком не предусмотрены средства для развертывания системы и весь проект окончится провалом. Кроме того, может выясниться, что развертывание комплекса в существующих условиях просто невозможно. Также может выясниться, что развертывание комплекса может быть слишком затратным мероприятием, стоимость которого превышает саму стоимость разработки.

Сопровождение ПО

Сопровождение ПО также следует предусмотреть еще на этапе проектирования, так как его стоимость может существенно повлиять на оценку прибыльности проекта. В ходе использования системы осуществляется информационная поддержка пользователей, возможно их обучение, устранение обнаруженных неисправностей, изменение функциональности и состава системы в соответствии с вновь появившимися требованиями заказчика. Если оговаривается соответствующий сервис, создается служба технической поддержки, которая осуществляет обслуживание комплекса, отвечает на возникающие вопросы. При обнаружении неисправностей или несоответствия заявленной функциональности производится замена соответствующих модулей вновь разработанными. В случае, если заказчик решает изменить функциональность системы, производится ее модернизация.

Лекция

Документация ПО

Стандарты ЕСПД в основном охватывают ту часть документации, которая создается в процессе разработки ПО. Следует отметить, что стандарты ЕСПД (ГОСТ 19) носят рекомендательный характер. Дело в том, что в соответствии с Законом РФ «О стандартизации» эти стандарты становятся обязательными на контрактной основе – то есть при ссылке на них в договоре на разработку (поставку) ПО.

Надо сказать, что наряду с комплексом ЕСПД официальная нормативная база РФ в области документирования ПО и в смежных областях включает ряд перспективных стандартов (отечественного, межгосударственного и международного уровней), например, Международный стандарт ISO/IEC 12207:1995-08-01 на организацию жизненного цикла ПО.

Перечень документов ЕСПД

- ГОСТ 19.001-77 ЕСПД. Общие положения.
- ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов.
- ГОСТ 19.102-77 ЕСПД. Стадии разработки.
- ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных документов.
- ГОСТ 19.104-78 ЕСПД. Основные надписи.
- ГОСТ 19.105-78 ЕСПД. Общие требования к программным документам.
- ГОСТ 19.106-78 ЕСПД. Общие требования к программным документам, выполненным печатным способом.
- ГОСТ 19.201-78 ЕСПД. Техническое задание. Требования к содержанию и оформлению.
- ГОСТ 19.202-78 ЕСПД. Спецификация. Требования к содержанию и оформлению.
- ГОСТ 19.301-79 ЕСПД. Порядок и методика испытаний.
- ГОСТ 19.401-78 ЕСПД. Текст программы. Требования к содержанию и оформлению.

- ГОСТ 19.402-78 ЕСПД. Описание программы.
- ГОСТ 19.404-79 ЕСПД. Пояснительная записка. Требования к содержанию и оформлению.
- ГОСТ 19.501-78 ЕСПД. Формуляр. Требования к содержанию и оформлению.
- ГОСТ 19.502-78 ЕСПД. Описание применения. Требования к содержанию и оформлению.
- ГОСТ 19.503-79 ЕСПД. Руководство системного программиста. Требования к содержанию и оформлению.
- ГОСТ 19.504-79 ЕСПД. Руководство программиста.
- ГОСТ 19.505-79 ЕСПД. Руководство оператора.
- ГОСТ 19.506-79 ЕСПД. Описание языка.
- ГОСТ 19.508-79 ЕСПД. Руководство по техническому обслуживанию. Требования к содержанию и оформлению.
- ГОСТ 19.604-78 ЕСПД. Правила внесения изменений в программные документы, выполняемые печатным способом.
- ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.
- ГОСТ 19.781-90. Обеспечение систем обработки информации программное.

Наряду с ЕСПД на межгосударственном уровне действуют еще два стандарта, также относящихся к документированию ПО и принятых не так давно, как большая часть ГОСТ ЕСПД.

- ГОСТ 19781-90 Обеспечение систем обработки информации программное. Термины и определения.
- Разработан взамен ГОСТ 19.781-83 и ГОСТ 19.004-80 и устанавливает термины и определения понятий в области ПО, систем обработки данных.
- ГОСТ 28388-89 Системы обработки информации. Документы на магнитных носителях данных. Порядок выполнения и обращения.
- Распространяется не только на программные, но и на конструкторские, технологические и другие документы, выполняемые на магнитных носителях.
- ГОСТ Р ИСО/МЭК 9294-93. Информационная технология. Руководство по управлению документированием ПО.
- ГОСТ Р ИСО/МЭК 9126-93. Информационная технология. Оценка программной продукции, характеристика качества и руководство по их применению.

- ГОСТ Р ИСО/МЭК 8631-94. Информационная технология. Программные конструктивы и условные обозначения для их представления
- ГОСТ Р ИСО/МЭК 8631:1994. Информационная технология. Пакеты программных средств. Требования к качеству и испытания.
- ГОСТ Р ИСО/МЭК 12207-99. Процессы жизненного цикла программных средств.
- ISO 12207:1995. (ГОСТ Р-1999). ИТ. Процессы жизненного цикла программных средств.
- ISO 15271:1998. (ГОСТ Р-2002). ИТ. Руководство по применению ISO 12207.
- ISO 16326:1999. (ГОСТ Р-2002). ИТ. Руководство по применению ISO 12207 при административном управлении проектами.
- ISO 9126:1991. (ГОСТ-1993). ИТ. Оценка программного продукта. Характеристики качества и руководство по их применению.
- ISO 12119:1994. (ГОСТ Р-2000). ИТ. Требование к качеству и тестирование.
- ISO 13210:1994. (ГОСТ Р-2000). ИТ. Методы тестирования для измерения соответствия стандартам POSIX.

Стандарт ISO 9000

Это целая группа стандартов, предъявляющая требования к процессу разработки, выпуска и поддержки продуктов (не обязательно программных систем) обеспечивающие их качество.

Первый вариант международного стандарта 9000-й серии был опубликован Международной организацией стандартов (ISO) в 1987 г., после чего он два раза пересматривался. Сегодня действующий вариант — ISO 9001:2008. Существенным отличием новой редакции стандарта от предыдущих является требование непрерывного улучшения качества и подход к описанию предприятия на основе бизнес-процессов.

ISO 9001:2008 устанавливает требования к системе менеджмента качества, которые предназначены для внутреннего применения организациями в целях сертификации или заключения контрактов. Он направлен на обеспечение эффективности системы менеджмента качества и формулирует минимальный набор условий, которым она должна удовлетворять для выполнения установленных требований к продукции. Соответствие стандарту ISO 9001:2008 может также использоваться организацией для демонстрации своей способности удовлетворить нужды и чаяния потребителей.

ISO 9004:2009 предоставляет методическую помощь по более широкому спектру целей системы менеджмента качества, нежели это делает ISO 9001:2008, особенно в отношении постоянного улучшения деятельности организации и повышения эффективности, а также ее результативности. Назначение обновленного ISO 9004 - предоставление методической помощи руководству по внедрению и применению системы менеджмента качества для совершенствования работы организации в целом. Эти методические указания охватывают создание, функционирование, поддержание в рабочем состоянии и постоянное улучшение системы менеджмента качества.

Стандарт ISO 9001:1994 - Система качества. Модель для обеспечения качества при проектировании, разработке, производстве, монтаже и обслуживании - предлагает оценивать способность поставщика не только успешно проектировать изделия, но также производить их и испытывать. В стандарте подробно изложены 20 групп требований к системе качества и ее компонентам, рубрикация которых используется и в других документах. Определена ответственность руководства за политику и организацию в области качества, представлены общие требования к управлению проектированием, документацией и проведением испытаний. Сформулированы правила регистрации данных о качестве, контроле и корректировках качества продукции, о подготовке специалистов в области качества.

Система качества

Функционирование и применение системы качества на предприятии - разработчике ПС проверяется путем аудита наличия комплекта документов и их непосредственного использования.

В процессе проверки системы качества анализируется наличие планов и методик проведения внутренних проверок и документально оформленных их результатов, обеспечивающих:

- выявление причин несоответствий продукций и корректирующих действий, предупреждающих повторное возникновение дефектов;
- анализ всех процессов, рабочих операций, отклонений, протоколов качества, отчетов об использовании продукции и рекламаций потребителя с целью выявления и устранения возможных причин дефектов продукции;
- проведение предупреждающих действий для предотвращения дефектов на том уровне, который соответствует реальному риску;
- проведение контроля с целью проверки реализации и эффективности корректирующих действий.

Облегченные методики разработки ПО

Классические методики разработки ПО рассчитаны на то, что известно, какой объем работы за какое время и какими силами необходимо выполнить. Однако зачастую требования к проекту могут быть сформулированы неточно или нечетко, неизвестны условия выполнения проекта, методики выполнения проекта являются новыми в данной индустрии или для группы разработчиков. То есть имеется целый ряд факторов, наличие которых не позволяет указать сроки, бюджет или объем проекта. В таких случаях используются методики более гибкой разработки требуемого ПО. В ряде случаев классические и новые методики могут комбинироваться. Так, например, для новых работ может быть составлен план проведения исследования, далее заключается договор на создание исследовательского прототипа системы, позже система дорабатывается, проводится е внедрение и опытная эксплуатация. По окончании всех этих этапов система внедряется в полном объеме. С одной стороны каждый из этапов разрабатывается классическим образом – с фиксированными требованиями, сроками и бюджетом. С другой стороны, подобная разработка осуществляется по спирали – общие сроки проекта, его бюджет и средства заранее не определены. Подобные ситуации часто возникают при работе с «квалифицированным» заказчиком, который осознает состояние дел в отрасли, представляет себе сложность и инновационность проблемы.

Здесь мы подробно рассмотрим три методики разработки ПО: спиральная разработка, экстремальное программирование и технология SCRUM.

Спиральная модель разработки

Спиральная модель разработки ПО построена следующим образом. На начальных этапах разработки формируются основные требования к разрабатываемой системе, ее базовая функциональность. Далее создается прототип системы, при работе с которым заказчик может выявить ряд нюансов, скорректировать свои требования, оценить сложность проекта. Далее функциональность системы изменяется и расширяется на основе вновь появившихся знаний о системе. В ходе работы над проектом проводится несколько таких итераций. В идеале – столько, сколько необходимо для его успешного завершения.

На каждой итерации рекомендуется придерживаться следующих шагов.

- Формирование видения проблемы. В ходе встречи с заказчиком выясняется, что именно заказчик желает получить на данный момент времени.
- Расстановка приоритетов. Требования, предъявляемые заказчиком к системе могут оказаться противоречивыми. Кроме того, может оказаться затруднительно реализовать сразу всю намеченную функциональность за один этап.
- Согласование временных рамок проекта. На данном этапе по составленному объему работ определяется общее время, необходимое на выполнение этапа. Часто при фиксированных сроках оговаривается, какую часть работ разработчик сумеет выполнить в соответствии с расставленными приоритетами.
- Определение архитектуры и ядра будущей системы. Ядро системы здесь – это законченный работающий вариант системы с урезанной функциональностью. С заказчиком оговаривают особенности функционирования системы в общем, а также особенности работы основных (наиболее важных) фрагментов. Далее определяется функциональность подсистем с более низкими приоритетами.
- Формирование плана выполнения итерации. В соответствии с оговоренными сроками и функциональностью производится планирование работ в рамках итерации.
- Разработка системы в соответствии с планом. На данном этапе производится создание самой системы. При этом в случаях отставание по срокам, сокращения финансирования или в других критических ситуациях, в соответствии с оговоренными приоритетами могут отбрасываться функции с низкими приоритетами.

Экстремальное программирование

Экстремальное программирование работает только в случае, если вы имеете небольшую команду профессионалов (не больше 10 человек), в которой отлажен механизм взаимодействия. Кроме того, каждый из сотрудников должен уметь переключаться на любой из видов работ, выполняемых вашей командой.

Суть экстремального программирования состоит в обобщении отдельных этапов разработки на весь процесс. Это означает, что в ходе выполнения всего проекта выполняется его проектирование, уточнение требований, разработка, тестирование и отладка.

В основе экстремального программирования лежат 12 принципов. В сущности, экстремальное программирование может остаться таковым, даже если при реализации отбросить некоторые (но не большинство) из этих принципов. С другой стороны, применение одного-двух принципов не сделает разработку экстремальной.

При таком подходе резко возрастает роль менеджера, отвечающего за проект. Он должен уметь организовать работу таким образом, чтобы выполнялись все указанные выше принципы.

Экстремальное программирование рассчитано на создание не очень больших систем, для которых ясны пути их реализации.

Принципы экстремального программирования

1. Простота проектирования – сдаваемая система будет наиболее простой системой, отвечающей требованиям заказчика – в нее не закладывается возможностей для развития.
2. Выбор приоритетов и управление рисками. На основе некоторых оценок принимается решение, какую функциональность следует сделать немедленно, а какую можно отсрочить.
3. Непрерывная интеграция – сборка продукта производится каждый день или по несколько раз в день.
4. Тестирование как методический прием – тестирование ведется непрерывно всеми членами команды. Программисты готовят тесты до того, как пишут программный код.
5. Небольшие релизы – разработчики выпускают работоспособные версии программы как можно чаще.
6. Выделенный пользователь – в состав команды входит представитель заказчика, отвечающий за проверку функциональности системы, ответы на вопросы о пожеланиях заказчика, выбор тех или иных решений, расстановку приоритетов.
7. Переоценка потребностей – программисты постоянно улучшают структуру проекта и кода (проводят рефакторинг), стараясь избегать повторной реализации сходного кода. При этом разделяют стадии дизайна и рефакторинга. Просмотр кода на предмет его улучшения производится только над стабилизированным кодом.
8. Взаимодействие – каждый из сотрудников должен уметь синхронизировать свою работу с действиями других. При этом каждый должен понимать структуру всего проекта.
9. Система имен – в ходе разработки используется единая терминология.
10. Парная разработка – программисты пишут код парами, вплоть до работы пары за одним компьютером. Пока один пишет код, второй работает над архитектурой.
11. Коллективное владение – код не принадлежит никому из команды, каждый член команды может работать над любым фрагментом кода.
12. 40-часовая рабочая неделя – усталые программисты делают больше ошибок.

Методология SCRUM

Методология SCRUM основывается на проведении недлинных (порядка 30 рабочих дней) этапов, называемых спринтами. В начале спринта проводится общее собрание, в котором принимает участи представитель заказчика. В ходе этого собрания составляется бэклог спринта – список работ, которые необходимо выполнить в ходе данного спринта. При этом работы ранжируются и для них определяются требования. В ходе самого спринта никто не имеет права менять бэклог спринта. В случае необходимости спринт может быть прерван и новый спринт начнется с самого начала – с общего собрания.

Кроме бэклога на конкретный спринт ведется бэклог проекта в целом. За добавление и удаление функций, входящих в бэклог, отвечает только один человек в команде, хотя сами новые требования могут поступать из различных источников.

В ходе выполнения спринта проводятся ежедневные собрания. Основной задачей таких собраний является ответ на три основные вопроса – что я сделал, что я буду делать дальше и что мне мешало мне работать или повысить производительность? Задачей менеджеров является устранение любых препятствий, мешающих работе разработчиков.

Цель ежедневных встреч – убедиться, что работа выполняется правильно и наметить пути ее выполнения на будущее. Встречи длятся порядка 15 минут и не требуют от участников проекта значительных затрат. Игнорирование ежедневных встреч приводит к снижению темпов работ и может вызвать крах проекта.

Методика не предусматривает деления процесса разработки на фиксированные этапы и/или группы. В ходе собраний в начале спринта разработчики решают, какие работы им необходимы на следующем этапе. Сами работы при этом оцениваются в функциональных возможностях проекта. Тестирование и документирование при этом рекомендуется осуществлять параллельно с работами по программированию проекта.

Методика SCRUM в большей степени ориентирована на создание больших и сложных систем, для которых требования не могут быть определены изначально. При грамотной постановке работ повышение производительности программистов может составлять до 600%. Однако при этом следует учитывать, что одной из рекомендаций методологии SCRUM является отсутствие конечных сроков сдачи продукта и его бюджета, что существенно повышает шансы проекта на успех.

Персональный процесс разработки

До сих пор мы рассматривали процесс разработки с точки зрения менеджера проекта. Однако современные методики предусматривают рекомендации для индивидуальных разработчиков, существенно повышающие их производительность и качество создаваемого ими кода.

Методикой, поясняющей как этого добиться, является персональный процесс разработки программного обеспечения – PSP. Методика PSP предлагает 7 этапов, проходя которые программист может улучшить собственный процесс.

В целом при работе по методике PSP программист проходит несколько стадий. При обучении любой деятельности в самом начале обучения человек бессознательно некомпетентен. Ситуация напоминает диалог. «Ты на скрипке играть умеешь» - «Не знаю, не пробовал». Человек не осознает свои возможности, которые чаще всего бывают существенно ниже необходимого уровня. После первых отрицательных результатов человек начинает осознавать собственный уровень, то есть становится сознательно некомпетентным. После долгих тренировок, решения уже практических задач, человек начинает осознавать в какой области он обладает знаниями, а в какой нет, то есть становится сознательно компетентным. Через некоторое время человек достигает вершин мастерства.

Задачей PSP является перевести программиста со второго уровня на третий и позволить ему как можно дольше оставаться на этом уровне для того, чтобы четко осознавать перспективы и направления собственного развития.

Этапы PSP

Этап PSP 0. На данном этапе программист **учится составлять общий план проекта**, оценивать время на каждый небольшой этап разработки, учится больше внимания обращать на тестирование разработанных модулей. Данный этап предназначен для того, чтобы приучить программиста документировать результаты свой работы и составлять список работ, необходимых для реализации проекта.

Этап PSP 0.1. На данном этапе программист **учится измерять объем работ** – строк кода программы в целом и ее отдельных модулей, страниц документации и т.д. Для каждой выполняемой задачи программист описывает объем работ, который ему пришлось выполнить для ее реализации, время которое он потратил на выполнение этого объема работ.

Этап PSP 1. На данном этапе программист **учится оценивать размер будущей программы**. При этом следует различать новый код (добавленный к существующему объекту, кодом нового объекта, измененным кодом объекта), повторно используемый код, базовый код прежней версии. При этом также следует учитывать, что разный код вносит разный вклад в итоговую оценку. Так, например, код, написанный на C, C++ и Visual Basic содержит обычно больше ошибок, чем код, написанный на Pascal.

Этапы PSP

Этап PSP 1.1. На данном этапе программист, на основе полученных количественных данных, **учится оценивать рабочее время**, которое потребуется для выполнения некоторого объема работ. Результатом является умение составлять календарный план работ. Календарный план составляется в следующем порядке: оценка требований продукта, основные модули продукта, оценка объема модулей, оценка времени, требующегося на реализацию модулей, составление графика работ. В дальнейшем программист ведет график реального выполнения работ, отслеживая отклонения запланированного графика от реального.

Этап PSP 2. Данный этап **учит программиста управлять качеством своего кода**. Программист пытается предсказать количество ошибок, которые могут появиться в его коде. В ходе разработки программист сверяется с плановыми числами и при нахождении расхождений пытается выявить их причину. Основным методом для выявления ошибок является «метод пристального взгляда» – тщательное изучение собственного кода после того, как он был написан и прошел компиляцию, однако перед тем, как начнется его тестирование самим программистом (хотя данный метод можно применять и до запуска компилятора). Дело в том, что компилятор пропускает порядка 10% потенциальных ошибок – отсутствие break в операторе switch, strlen(str+1) и прочие. При этом время, необходимое на обнаружение ошибки в ходе финального тестирования, выше времени обнаружения ошибки в ходе просмотра кода на порядки. Если в ходе обзора кода ошибка находится за 5-20 минут, то в ходе тестирования на ее обнаружение уходит 15-30 минут, а в ходе комплексного тестирования – порядка 8 часов. Число ошибок на 1000 строк кода снижается примерно в 5 раз для ошибок, обнаруживаемых компилятором, и в 3 раза – обнаруживаемых при тестировании.

Этапы PSP

Этап PSP 2. Здесь программист **учится качественно проектировать программу в целом**. На предыдущих этапах при заданных требованиях следовало разделить программу на блоки и оценить их с точки зрения реализации. На основе полученных знаний программист получает возможность анализировать требования к программе, создавать спецификации проекта, выполнять высокоуровневое проектирование.

Этап PSP 3. Посвящен **совершенствованию персонального процесса разработки**. Здесь программист ставит себе некоторую цель по повышению своей производительности или улучшению качества кода. Далее определяются методы достижения этой цели, проводятся замеры характеристик работы, полученный характеристики анализируются и на их основе программист вырабатывает рекомендации по улучшению.

Уровни зрелости компании

Для оценки эффективности производства программными продуктами используют уровни зрелости компании. В соответствии с рекомендациями Software engineering institute существует пять уровней зрелости.

1. Начальный – отсутствует статистический контроль и управление, планирование и управление рабочим временем, затратами.

2. Повторяющиеся процессы – в организации проводится повторяющийся (регулярный) статистический контроль за общим состоянием проекта, его сроками, степенью соответствия проекта требованиям, стоимостью.

3. Определенные процессы – на основании имеющейся информации об уже завершенных проектах компания получает возможность более точно прогнозировать затраты на определенные процессы, пути их выполнение.

4. Управляемые процессы – компания проводит регулярные замеры и анализ производительности, в результате чего любой проект в всегда является полностью управляемым.

5. Оптимизированные процессы – все процессы, протекающие в компании, исследованы настолько, что появляется возможность произвести их оптимизацию.

1 Этап

Практически все компании, начинающие свою деятельность находятся на первом уровне зрелости, зачастую называемом хаотическим. На этом уровне организации практически полностью **отсутствуют детальные планы разработки проекта, оценки стоимости и рабочего времени**. Использование инструментов является минимальным, стандартные методики и технологии больше частью игнорируются. Отсутствует контроль за уже разработанным кодом, не ведется список имеющихся задач и проблем. Основным решением в любой кризисной ситуации является увеличение нагрузки на членов команды.

Выходом из такой ситуации является введение управления проектом. Оно подразумевает предварительную выработку требований проекту, промежуточных планов выполнения, подбор инструментария, оптимально подходящего для работы над проектом. В ходе самого проекта должны выполняться выработанные промежуточные сроки либо производиться своевременная переоценка плана. Уровень руководства должен обладать актуальной информацией, проводить регулярные проверки, быть в курсе прогресса работ и возникающих проблем.

Кроме того, **необходимо ввести постоянный контроль за качеством**. Если проект выполняется в сроки, но не соответствует требованиям или не является функциональным, можно говорить о кризисной ситуации. Это связано с тем, что на доработку проекта потребуется дополнительное время, не предусмотренное планом и объемом финансирования. На заключительной стадии проекта несоответствие требованиям может привести также к тому, что клиент не примет выполненную работу.

2 Этап

Повторяющиеся процессы позволяют наладить управление процессом разработки, сделать его более предсказуемым. Кризисы в таких компаниях встречаются реже и причины их возникновения чаще находятся вне компании. Компании, находящиеся на уровне повторяющихся процессов, обычно состоят из профессионалов, имеющих опыт разработки, обладающих знаниями о процессах проектирования и разработки. Они имеют представление о том, какие затраты необходимы для реализации отдельных фрагментов проекта, какие инструменты лучше применять, как организовать ход работ над проектом.

Однако на данном этапе **отсутствует полная точность**. Это может быть связано с тем, что команда работает в новой для нее области и не представляет всех нюансов, которые могут возникнуть. Такая ситуация возникает в случае, когда команда не имеет достаточного опыта либо вторгается в новую область деятельности. Несмотря на это, проводится постоянное управление проектом, хотя сроки выполнения проекта могут корректироваться.

Для того, чтобы перейти на следующий уровень зрелости компания должна определить группы процессов, отделив процессы, в разработке которых команда имеет опыт, от тех, с которыми приходится сталкиваться впервые. Далее компания должна выбрать некоторую архитектуру разработки программного обеспечения, выбрать методики и методологии, наиболее подходящие для работы над проектом.

3,4,5 этапы

Переход к определенным процессам позволяет осуществлять разработку твердо представляя себе порядок, объем и сложность работ, уровень затрат. **Команда представляет себе возможные кризисные ситуации и обладает знаниями о методах выхода из них.** Сами кризисные ситуации становятся реже, так как детальная проработка проблемы позволяет принять превентивные меры по их устранению.

Но данные и управление на третьем уровне все еще остаются качественными, а не количественными.

Для перехода на четвертый уровень необходимо собирать все статистические данные по уже сделанным процессам и применять их в дальнейшем на этапе проектирования. При этом сбор статистической информации поручается руководящему звену компании, а в некоторых случаях – специально уполномоченным сотрудникам. Должны проводиться специальные оценки эффективности выполнения различных этапов проверки с последующим анализом и коррекцией планов при выполнении аналогичных работ.

На пятом уровне компания получает возможность оптимизировать проходящие в ней процессы. При этом твердо закреплены качественные и количественные характеристики, по которым проводятся измерения и оценки, сбор информации производится автоматически или при помощи автоматизированных средств, результаты анализа данных динамично используются в задачах планирования и прогнозирования. Данные на данном уровне собираются в единые хранилища информации, которые обеспечивают не только контроль за изменениями и документирование кода, но и поиск прецедентов, шаблонов, стандартных алгоритмов и решений. На данном уровне автоматизируется не только сбор

Лекция. Инструментальные средства разработки ПО. История

Несмотря на развитие инструментальных средств до середины 70-х годов не наблюдалось роста производительности труда программистов. Напротив, из-за ужесточения требований и нелинейного роста сложности программного обеспечения при увеличении его размеров уменьшалась производительность труда. Постоянно срывались сроки разработки. ПО становилось дороже, а качество непредсказуемо. И самое неприятное, что не срабатывали испытанные методы исправления ситуации (предоставление дополнительных ресурсов — материальных и человеческих). Возникшее положение получило название «кризиса программного обеспечения».

Новый этап развития программного обеспечения был подготовлен открытиями 70-х годов в области технологии программирования и может датироваться публикацией работы Боэма «Программная инженерия» в 1976 году. В этой работе вводилось понятие жизненного цикла ПО и обосновывалось, что основные затраты приводятся не на разработку программ и не на их отладку, а на сопровождение. В истории развития ПО выделяют 3 этапа.

История. 1 этап

На первом этапе основные затраты были связаны с кодированием программ, реализующих известные алгоритмы вычисления. Это привело к разработке средств автоматизации кодирования (автокодов), что уменьшило затраты примерно на порядок. Засчет автоматизации рутинных операций, устранения или своевременного обнаружения ошибок. Принятый в настоящее время термин «Ассемблер» является неудачным, так как не раскрывает отличительные черты данного класса инструментальных средств. Например, команда на ассемблере **met clc a,b** определяет не алгоритмическое действие, а заполнение некоторой области памяти в ЭВМ, в данном случает 6 байт. И только от программиста зависит как будет использоваться эта область, в качестве машинной команды или в качестве данных. Обычно ассемблеры являются машинно-ориентированным языками низкого уровня, то есть подразумевает один оператор языка одна машинная команда. Но почти в каждом из них имеются операторы управления адресацией размещения программы, форматы и составы распечаток, что не соответствует машинным командам. Автоматизация кодирования подталкивает к повышению уровня языка включением в него макросредств, что приводит к уменьшению затрат. Для ассемблера ЕС ЭВМ существовали комплекты макрокоманд структурного программирования. Были накоплены сотни системных макрокоманд с нетривиальными алгоритмами проверки корректности операндов и подстановки с макро расширениями. В ассемблерах впервые появились средства раздельной компиляции, в которой были первым шагом виртуализация ресурсов. Это привело к разработке специальных промежуточных языковых к появлению новых инструментальных программ, загрузчиков и компановщиков.

История. 2 этап

Второй этап развития программных средств характеризуется увеличением размеров программ и увеличением затрат на преодоление разрыва между понятиями проблемных областей и машинных языков. Для преодоления разрыва создаются языки высокого уровня (АЛГОЛ-60 и ПЛ/1). Универсальность этих языков была относительна, но они были широко распространены. В то время существовала иллюзия — вера в неограниченную возможность с повышением уровня языка. Было разработано около 3000 языков программирования. Многие из них использовались только авторами и мало отличались друг от друга. Несмотря на мнение о нецелесообразности разработки новых языков, наблюдается дефицит специализированных языков для узких проблемных областей.
До сих пор Lisp является языком программирования для задач искусственного интеллекта. Fortran, сильно изменившийся в наше время, применяется для решения вычислительных задач. В тот период были изобретены и опробованы все основные типы данных, операции над ними, управляющие структуры, способы изображения в программах. Развитие не было планомерным, далеко не сразу осозналось значение найденных решений.

История. 3 этап

Третий этап бы связан с обобщением и интеграцией найденных решений в больших языков программирования. Движущей силой процессов было существенное изменение затрат в результате резкого увеличения размеров систем программного обеспечения до сотен тысяч и даже миллионов строк исходного текста. А также коллективный характер работ над программным обеспечением и переход к товарному производству. Были разработаны развитые операционные системы, первые СУБД, многочисленные системы автоматизации документирования, отслеживания модификаций и сборки версий программного обеспечения.

Инструментальные средства разработки ПО

В процессе разработки программного обеспечения используется большое количество самого разностороннего ПО:

Необходимые

- компиляторы и ассемблеры;
- редакторы текстов;
- компоновщики или редакторы связей (linkers).

Часто используемые

- утилиты автоматической сборки проекта;
- отладчики;
- программы создания файлов помощи (документации).
- программы создания инсталляторов.

Специализированные

- дизассемблеры;
- Декомпиляторы.

Интегрированные среды разработки

- Интегрированные среды разработки содержат большую часть из приведенных выше программ и позволяют упростить процесс создания приложений.

Текстовые редакторы

Не имея текстового редактора, мы не сможем редактировать исходный текст на языке программирования. Важно понимать, что в этом контексте под текстовым редактором понимается не что-то вроде Microsoft Word с его многообразием функций для форматирования текста, таблиц и т.п.. Текстовый редактор для программирования должен обеспечивать немного другой функционал:

- возможность сохранения документов в требуемом формате (поддержка нужных расширений и т.п.);
- возможность подсветки синтаксиса (выделение ключевых слов, констант и т. п.);
- возможность копирования, вставки, замены фрагментов текста;
- возможность автодополнения вводимого текста, отображения вариантов для вызываемых функций и т.п.;
- возможности автоформатирования вводимого текста (автоматически создавать отступы в теле функции и т.п.).

Поскольку работа с текстовым редактором занимает значительную часть всего времени разработки, правильный выбор текстового редактора может существенно ускорить или замедлить процесс редактирования исходных текстов. Кроме того, практически все интегрированные среды разработки используют свой редактор. Все эти редакторы имеют общие черты, но есть и отличия, по которым и формируется окончательное мнение человека о среде разработки в целом.

Если не рассматривать консольные редакторы (которые все еще популярны под операционными системами семейства UNIX), то можно с уверенностью сказать, что все редакторы, поддерживающие GUI (Graphical user unterface, графический интерфейс пользователя) имеют схожий пользовательский интерфейс.

Компилятор

Итак, у нас есть исходный текст, написанный на языке программирования. Он составлен в соответствиями с правилами языка и готов к дальнейшей обработке. Каким образом компьютер может выполнить этот исходный текст? Тут нам на помощь приходит программа, именуемая интерпретатор или компилятор. Обе эти программы выполняют схожие задачи (преобразование исходных текстов в машинный код), но подходят к ним по-разному.

Компилятор

Компилятор преобразует сразу весь исходный текст. На выходе, как правило, получается один или несколько файлов объектного кода. Будем считать, что объектный код - что-то вроде полуфабриката, который практически готов к употреблению процессором. В некоторых языках программирования, процесс компиляции может управляться так называемыми директивами компиляции - специальными указаниями о том, как компилировать отдельные блоки текста. В общей сложности, компиляция программы состоит из следующих этапов:

- Лексический анализ. На этом этапе весь текст исходного файла преобразуется в последовательность лексем.
- Синтаксический (грамматический) анализ. Последовательность лексем преобразуется в дерево разбора.
- Семантический анализ. Дерево разбора обрабатывается с целью установления его смысла — например, привязка идентификаторов к их декларациям, типам, проверка совместимости, определение типов выражений и т. д. Результат обычно называется «промежуточным представлением».
- Оптимизация. Выполняется удаление лишних конструкций и упрощение кода с сохранением его смысла. Оптимизация может быть на разных уровнях и этапах. Так же, оптимизация может заключаться в использовании группы команд более современного процессора.
- Генерация кода. Из промежуточного представления порождается код на целевом языке.

Интерпретатор

Интерпретатор тоже преобразует текст на языке высокого уровня, но делает это построчно. Как только первая строка преобразована - она тут же запускается на выполнение. У такого подхода есть как достоинства, так и недостатки.

Достоинства интерпретаторов:

- Бóльшая переносимость интерпретируемых программ — программа будет работать на любой платформе, на которой есть соответствующий интерпретатор.
- Как правило, более совершенные и наглядные средства диагностики ошибок в исходных кодах.
- Упрощение отладки исходных кодов программ.
- Меньшие размеры кода по сравнению с машинным кодом, полученным после обычных компиляторов.

Недостатки интерпретаторов:

- Интерпретируемая программа не может выполняться отдельно без программы-интерпретатора. Сам интерпретатор при этом может быть очень компактным.
- Интерпретируемая программа выполняется медленнее, поскольку промежуточный анализ исходного кода и планирование его выполнения требуют дополнительного времени в сравнении с непосредственным исполнением машинного кода, в который мог бы быть скомпилирован исходный код.
- Практически отсутствует оптимизация кода, что приводит к дополнительным потерям в скорости работы интерпретируемых программ.

Компоновщик

Компоновщик, известный так же как редактор связей, (linker, link editor) занимается созданием готового исполняемого файла из «кусочков» объектного кода. Выше, в разделе компиляции мы рассматривали ситуацию, когда проще выполнить компиляцию по частям. Если был применен такой подход, то окончательную сборку выполняет компоновщик. Как-правило, компоновщик является частью компилятора.

Для связывания модулей компоновщик использует таблицы имён, созданные компилятором в каждом из объектных модулей. Такие имена могут быть двух типов:

- Определённые или экспортируемые имена — функции и переменные, определённые в данном модуле и предоставляемые для использования другим модулям;
- Неопределённые или импортируемые имена — функции и переменные, на которые ссылается модуль, но не определяет их внутри себя.

Работа компоновщика заключается в том, чтобы определить и связать ссылки на неопределённые имена в каждом модуле. Для каждого импортируемого имени находится его определение в других модулях и после этого упоминание имени заменяется на его адрес.

Отладчик

Отладчик или дебаггер является модулем среды разработки или отдельным приложением, предназначенным для поиска ошибок в программе. Отладчик позволяет выполнять пошаговую трассировку, отслеживать, устанавливать или изменять значения переменных в процессе выполнения программы, устанавливать и удалять контрольные точки или условия остановки и т. д.

Список отладчиков:

- 1) AQtime — коммерческий отладчик для приложений, созданных для .NET Framework версии 1.0, 1.1, 2.0, 3.0, 3.5 (включая ASP.NET приложения), а также для Windows 32- и 64-битных приложений.
- 2) DTrace — фреймворк динамической трассировки для Solaris, OpenSolaris, FreeBSD, Mac OS X и QNX.
- 3) Electric Fence — отладчик памяти.
- 4) GNU Debugger (GDB) — отладчик программ от проекта GNU.
- 5) IDA — мощный дизассемблер и низкоуровневый отладчик для операционных систем семейства Windows и Linux.
- 6) Microsoft Visual Studio — среда разработки программного обеспечения, включающая средства отладки от корпорации Microsoft.
- 7) OllyDbg — бесплатный низкоуровневый отладчик для операционных систем семейства Windows.
- 8) SoftICE — низкоуровневый отладчик для операционных систем семейства Windows.
- 9) Sun Studio — среда разработки программного обеспечения, включающая отладчик dbx для ОС Solaris и Linux, от корпорации Sun Microsystems.
- 10) Dr. Watson — стандартный отладчик Windows, позволяет создавать дампы памяти.
- 11) TotalView — один из коммерческих отладчиков для UNIX.
- 12) WinDbg — бесплатный отладчик от корпорации Microsoft.

Генератор документации

Генератор документации — программа или пакет программ, позволяющая получать документацию, предназначенную для программистов (документация на API) и/или для конечных пользователей системы, по особым образом комментированному исходному коду и, в некоторых случаях, по исполняемым модулям (полученным на выходе компилятора).

Обычно генератор анализирует исходный код программы, выделяя синтаксические конструкции, соответствующие значимым объектам программы (типам, классам и их членам/свойствам/методам, процедурам/функциям и т. п.). В ходе анализа также используется мета-информация об объектах программы, представленная в виде документирующих комментариев. На основе всей собранной информации формируется готовая документация, как правило, в одном из общепринятых форматов — HTML, HTMLHelp, PDF, RTF и других.

Инструментальные средства разработки клиент-серверных приложений

Любое клиент-серверное приложение состоит из клиентского и серверного приложений. Соответственно этому имеются инструментальные среды разработки клиентской части и серверной. В качестве первых обычно выступают интегрированные среды разработки, ИСР (Integrated Development Environment, IDE). В качестве вторых - системы управления базами данных, СУБД.

Инструментальные средства для разработки клиентской части

Клиентской называется часть приложения, с которой напрямую взаимодействует конечный пользователь. Это может быть либо приобретенное компанией серийное коммерческое программное обеспечение, либо прикладная программа, разработанная внутри компании с помощью инструментальных средств третьих фирм.

Для того, чтобы воспользоваться многочисленными новейшими инструментальными средствами, предназначенными для создания клиентской части приложений, которые доступны сегодня на рынке программного обеспечения, разработчики должны уметь программировать на таких языках, как C++ и HTML, или на одном из множества других процедурных языков программирования, предназначенных для разработки Web-приложений. Раньше для разработки пользовательских корпоративных программ, работающих в основном в символьном режиме, использовались такие языки программирования, как ANSI C, COBOL, FORTRAN и Pascal. Сегодня большинство вновь разрабатываемых клиентских прикладных программ является GUI-приложениями - они содержат графический интерфейс пользователя. Большинство из доступных сегодня инструментальных средств являются дружественными по отношению к пользователю и объектно-ориентированными. Наиболее популярными средствами для создания Web-приложений являются C++-Builder и IntraBuilder фирмы Borland, а также Visual J++ и Visual C++ компании Microsoft. Другие популярные средства разработки корпоративных приложений для локальных вычислительных сетей - PowerBuilder компании Powersoft, Developer/2000 корпорации Oracle, Visual Basic компании Microsoft и Delphi фирмы Borland.

Инструментальные средства для разработки серверной части

Ядром любой прикладной программы является ее серверная часть. Именно здесь незаметно для конечного пользователя базы данных происходит вся основная работа. Серверная часть приложения включает сам сервер БД, источники данных, а также связующее программное обеспечение, с помощью которого приложение подключается к Web-серверу или удаленной базе данных в локальной сети (важнейшими серверами баз данных являются Oracle, Informix, Sybase,Microsoft SQL Server и Borland InterBase.)

Обычно это является первым шагом при подключении любого приложения к корпоративной среде предприятия или окружению Intemet. Сервер баз данных устанавливается в этом случае местным администратором БД, хорошо представляющим себе как нужды компании, так и требования, предъявляемые прикладной программой.

Подключением приложения называется процесс его реализации в доступном пользователям окружении.

Связующее программное обеспечение для подключаемого приложения включает Web-сервер и какое-нибудь инструментальное средство, предназначенное для соединения Web-сервера с сервером баз данных. Главным требованием в этом случае является наличие на Web-сервере прикладной программы, способной общаться с корпоративной базой данных.