Технология программирования

Лекция 2

Зариковская Наталья Вячеславовна,

доцент кафедры ЭМИС

6. Тестирование, обеспечение качества

- Три аксиомы одного из самых первых советских программистов с несколько необычной фамилией Шура-Бура.
- **Аксиома 1.** В каждой программе есть ошибка.
- **Аксиома 2.** Если в программе нет ошибок, значит, в исходном алгоритме есть ошибка.
- **Аксиома 3**. Если ни в программе, ни в алгоритме ошибок нет, то такая программа никому не нужна.

Еще 30 лет назад был популярен критерий полноты тестирования, при котором набор тестов гарантировал, что по каждому ребру графа управления программы исполнение программы хотя бы один раз пройдет.

Чтобы проверить цикл, нужно создать тест, гарантирующий прохождение цикла 0, 1 и 2 раза.

0 – вдруг условие входа в цикл сразу было ложным, 1 – нормальное прохождение, 2 – не столь очевидно.

Возможно, в конце цикла есть присваивание переменной, которая в начале цикла только читается. Понятно, что однократный проход по циклу не даст возможности проверки важного и весьма вероятного варианта.

Изложенные критерии носят только эвристический характер, но весьма полезны.

Таким образом, мы видим, что программирование и тестирование – совершенно разные типы деятельности, требующие разных исполнителей.

Тестирование, обеспечение качества

- Программирование – это конструктивный созидательный процесс, который требует высокой квалификации и определенного оптимизма (см. аксиомы Шуры-Буры).
- Тестирование – деструктивный процесс, который требует высокой дотошности, подозрительности, пессимистичности, но, вообще говоря, не требует высокой квалификации (мы не говорим о системных программистах, которые создают инструментальные средства автоматизации тестирования, а об обычных тестерах).
- В каждом коллективе есть люди с неуживчивым характером, любящие покритиковать коллег по работе. Цены им не будет в группе тестирования.
- Еще раз повторим, что тестирование нужно вовсе не для того, чтобы показать, **что программа работает правильно – это невозможно**.
- Тестирование – это процесс исполнения программы с целью нахождения ошибок.
- Хороший тест – это тест, на котором с большой вероятностью ошибка найдется

Организационные и технологические соображения относительно тестирования

- Тестирование (не считая начальных отладочных тестов) **не должно проводиться самими авторами программы**. Наблюдается определенная ≪замыленность≫ взгляда автора на программу, поэтому он и тесты будет готовить на те варианты, которые предусмотрел в программе.
- Должна быть конкуренция между программистами и тестировщиками, нужен дух соревнования: ≪Я все равно тебя поймаю≫ против ≪Врешь, не поймаешь≫.
- Необходимо уметь оценивать вероятность и примерное количество оставшихся ошибок. **Например,** известен метод оценки количества рыб в пруду. Ловят, скажем, 100 рыбок, помечают их и выпускают обратно в пруд. Затем снова ловят 100 рыбок. Если почти все они мечены, значит, в пруду примерно 100 рыбок и есть, если же меченых попалось мало, то рыб, скорее всего, больше, причем в той же пропорции, какова доля непомеченных из 100 вновь пойманных.
- Во многих фирмах применяют тот же прием: специальные люди вставляют некоторое количество разнообразных ошибок в программы и возвращают их на доработку авторам с требованием обнаружить ровно столько же ошибок. По тому, какая часть найденных ошибок относится к специально вставленным, а какая – к вновь найденным ошибкам, можно судить о числе оставшихся ошибок.
- Еще проще – поручить тестирование одной и той же программы двум группам тестировщиков и оценить процент ошибок, найденных обеими группами.
- Должно проверяться не только нормальное поведение программы, но и поведение в случае неправильных входных данных и других ошибочных ситуаций на предмет устойчивости программы, осмысленности сообщений об ошибках и т.д.

Всегда должен быть четко оговорен конечный результат тестирования.

- В идеале еще до начала работ или на одной из ранних стадий должен быть создан и согласован с

Тестирование по типам

Различают тестирование по типам:

- черный ящик (без просмотра исходного текста);
- белый ящик (с изучением исходного текста);

по объему:

- маленький тест, типа печати *«hello, world»*, чтобы понять, есть ли вообще о чем говорить;
- получасовое тестирование (американцы говорят ≪Тест на одну сигарету≫), обычно проверяют по одному тесту на каждую функцию программы перед серьезным тестированием;
- модульное тестирование;
- комплексное тестирование.

Особого упоминания заслуживают тестирование граничных значений входных данных, тесты на максимальный объем счета, проверка предположений об ошибках (≪Я бы сделал ошибку здесь≫).

Иногда применяется перекрестное чтение особо ответственных участков программы, когда одна группа разработчиков читает программы другой группы (inspection peer review). В США очень популярны Bugs festivals, когда незадолго перед выпуском системы фирма платит определенную сумму за каждую найденную ошибку. Список таких приемов можно расширить, но вряд ли их можно считать общеупотребительными.

Тестирование по типам

В зрелых программистских компаниях для каждого проекта ведется своя база данных ошибок, в которую вносится:

- кто нашел ошибку, дата;
- описание ошибки;
- модуль, в котором ошибка обнаружилась (возможно, это наведенная ошибка, может быть, она вызвана ошибкой в совсем другом модуле);
- версия продукта;
- статус ошибки:

open: найдена

fixed: исправлена

can't reproduce: невозможно воспроизвести

by design: ошибка проектировщиков

won t fix: это не ошибка (тестеру показалось)

postponed: сейчас исправить трудно, исправим в следующей версии

regression: исправленная ошибка появилась вновь

Тестирование по типам

- Важность (severity) ошибки

crash: все падает, полная потеря данных

major problem: падает частично, частичная потеря данных

minor problem: что-то не то, но данные не теряются

trivial: сейчас не стоит исправлять

- Приоритет ошибки:

highest: невозможно поставить продукт с такой ошибкой, не можем перейти к следующей версии

high: поставить не можем, но можем перейти к следующей версии

medium: можем и исправим

low: косметические улучшения – оставим на следующую версию.

Тестирование

- Особенно заметна ценность базы данных ошибок для больших и длительных проектов.
- Если идентификатор одной ошибки встречается десятки раз в различных письмах, недельных отчетах – это верный признак того, что требуется вмешательство руководства.
- Все руководители разных рангов знают на память записи из этой базы данных о текущих ошибках с высшей важностью и приоритетом.
- По завершении каждого проекта база данных ошибок внимательно анализируется. Интересно распределение ошибок по тому, кто их совершил, по времени исправления, кто чаще ошибки находит и т.д. Если в одну неделю разработчик 1 сделал 10 ошибок, а разработчик 2 – только 2, это еще ни о чем не говорит. Но если же за весь период разработки разработчик 1 сделал 100 ошибок, а разработчик 2 – только 10, то по этим цифрам можно уже судить об их квалификации. Можно также оценивать целые группы или, скажем, качество проектирования, а можно сделать какие-то выводы по мощности и надежности используемых инструментальных средств.
- На основе изучения записей в базе данных ошибок руководство проектом принимает решение о возможности выпуска продукта, например, продукт нельзя выпускать, если есть хотя бы одна ошибка с важностью «crash» или с приоритетом «highest/high». Возможна и такая стратегия, когда продукт выпустить обязательно надо, но времени на исправление основных ошибок нет (нужно помнить, что даже небольшие исправления могут повлечь за собой новые ошибки), тогда из продукта просто удаляют часть функций, в которых есть ошибки.

Тестирование

На самом деле, любая зрелая программистская компания имеет большую независимую группу оценки качества ПО (Quality Assurance или просто QA), функции которой намного шире, чем просто тестирование.

Для каждого продукта проверяется:

- полнота и корректность документации;
- корректность процедур установки и запуска;
- эргономичность использования;
- полнота тестирования.

QA должна играть роль придирчивого пользователя, но внутри компании. В США разработчики и QA сидят в разных концах коридора, не поощряется даже неформальное общение. В западных компаниях довольно существенную часть зарплаты (примерно 20-30%) составляет бонус, выплачиваемый в конце проекта и только в случае его успешного завершения. Так вот, если QA найдет слишком много ошибок разработчиков, те остаются без бонуса, но если QA принял разработку, а затем пользователи начнут жаловаться, то QA остается без бонуса, хотя жалобы связаны с ошибками разработчиков. Таким искусственным разделением ответственности и непоощрением дружеских отношений между разработчиками и QA владельцы компаний пытаются застраховаться от неудачи на рынке.

В США говорят, что каждому продукту дается только одна попытка выхода на рынок. Если пользователям по разным причинам продукт не понравился, репутация теряется навсегда. Поэтому ошибки так дорого стоят.

7. Групповая разработка, управление версиями

Рассмотрим пример. Коллектив разрабатывает ПО телефонных станций, которое включает в себя следующие компоненты:

- ОС реального времени;
- драйверы телефонного оборудования;
- функциональное ПО;
- программы рабочих мест операторов (РМО);
- БД конкретного экземпляра АТС.

Каждый компонент разрабатывается отдельным коллективом разработчиков, но все компоненты тесно связаны друг с другом. Более того, каждый компонент (кроме, пожалуй, драйверов) состоит из большого числа модулей, разрабатываемых разными специалистами. Таким образом, налицо проблема взаимодействия большого количества разработчиков, каждый из которых может находиться в своей фазе разработки, может внести в свою программу такие изменения, которые не позволят другому разработчику отлаживать его программу и т.д. К сожалению, даже в коллективах, в которых слово ≪технология≫ царит уже более 20 лет, нередки случаи, когда при выезде на объект БД не соответствует ФПО или версия РМО уже устарела.

Техническая вооруженность компании позволяет быстро справляться с этими трудностями, но свести проблемы к нулю можно лишь жесткими организационными мерами, прежде всего, требуется преодолеть многие советские привычки и особенности нашего менталитета.

7. Групповая разработка, управление версиями

- В устоявшихся коллективах обычно используется трехуровневая система версий.
- Разработчик действует в своем файловом пространстве (≪библиотека разработчика≫), делает там, что хочет, и ни с кем ничего не согласовывает. Когда группа разработчиков решает, что какие-то модули отлажены (или когда их руководитель решает, что ждать больше нельзя, сроки поджимают), исходные тексты модулей передаются в библиотеку тестирования.
- Система или какой-то ее компонент проверяется на всех существующих тестах, причем библиотека тестирования остается фиксированной, все найденные ошибки заносятся в базу данных ошибок и исправляются в библиотеках разработчиков. Когда все тесты пропущены, библиотека тестирования вместе со списком найденных ошибок копируется в библиотеку предъявления, а библиотека тестирования готова к приему новой версии из библиотек разработчиков.
- В **библиотеку предъявления перенос осуществляется сразу же, когда пропущены все тесты**, а вот **перенос в библиотеку тестирования – только по мере готовности группы разработчиков выдать новую версию системы**.
- В чем же состоит роль библиотеки предъявления? Дело в том, что большие системы создаются большими коллективами с иерархической системой подчинения. У руководителя любой группы есть начальник, а у самого большого начальника есть заказчик, словом, твой начальник всегда может потребовать у тебя версию для комплексирования системы на его уровне. Так вот, лучше отдать ему пусть немного устаревшую версию, но с известным списком ошибок (пусть на следующем уровне часть тестов, где используются ошибочные функции, пока не гоняют), чем более новую, ≪с пылу, с жару≫, но с непредсказуемым поведением. Итак, библиотека предъявления ждет своего часа, когда начальник ее затребует, а может и так случиться, что библиотека предъявления успеет несколько раз обновиться, пока ею заинтересуются.

Групповая разработка, управление версиями

- Описанная выше трехуровневая система версий является слишком крупноблочной. В современных технологиях используются специальные инструментальные средства, позволяющие фиксировать каждое изменение и откатываться к нужной точке, например, Visual Source Safe или PVCS. Обычно такие системы версионирования устанавливаются на сервере, разработчики играют роль клиентов (тем самым автоматически решаются технические проблемы, например, разрешение конфликтов при одновременном обращении, вопросы back-up и др.), а система предоставляет следующие методы:
- *Check-out:* взять текст на исправление;
- *Check-in*: вернуть обратно;
- *Undo check-in*: отмена всех изменений, сделанных во время последней сессии;
- *Get latest version*: берется один файл или файлы целого проекта в том состоянии, в котором их застал последний check-in.
- Последняя версия обычно хранится отдельно.
- Само собой разумеется, что в системе версионирования можно хранить исходные тексты программ, двоичные файлы, документацию и т.п.

8. Психология программирования

- Было замечено, что отладка модуля размером в одну страницу может быть не на проценты, а в разы проще отладки модуля размером в одну страницу и еще 4-5 строк на другой странице.
- Оказалось, что это связано с принципами организации человеческой памяти.
- Есть сверхоперативная память, связанная, в основном, со зрением. Эта память имеет очень быстрый доступ, но очень мала – 7-9 позиций. (есть мнение, что у профессиональных программистов эта память имеет 20-25 позиций.)
- Существенно больше оперативная память, в которой и происходит вся основная мыслительная деятельность, но данные в ней не могут храниться долго.
- Самая большая — долговременная память. Человеку непросто заложить туда данные, но хранятся они долго.
- С устройством памяти связан принцип ≪центрального≫ (в отличие от ≪периферического≫) зрения. Человек хорошо воспринимает какую-то точку и то, что ее окружает.
- Если при отладке программы автор должен обозревать больше, чем одну небольшую страницу текста, он не может полноценно воспринять программу.

«Листать вредно»

- Еще один важный принцип инженерной психологии также связан с устройством человеческой памяти – принцип возможно раннего обнаружения ошибок.
- Если программист написал программу и тут же заметил ошибку, то ее исправить сравнительно просто. Если же ему сообщают о найденной ошибке через полгода, ее исправление превращается в проблему. Именно по этой причине мы являемся сторонниками статических алгоритмических языков высокого уровня (АЯВУ, в которых транслятор в каждой точке программы ≪знает≫ виды обрабатываемых значений), а не динамических, в которых типы данных определяются во время счета. В динамических языках простое несовпадение типов может обнаружиться и через полгода, когда сложится ≪нужное расположение звезд на небе≫.
- В инженерную психологию как в науку поверили.

Пирамида Маслова

- Многое в поведении людей объясняет пирамида Маслова:
- Диаграмма Маслова носит совершенно тривиальный характер – если человеку нечего есть и негде жить, то что ему до высоких материй?
- Но совокупное понимание человеческих потребностей и их выстроенная последовательность очень важны. В каком-то смысле каждый программист проходит весь путь от простого зарабатывания денег до понимания того, как важно быть членом команды, признания товарищей, достижения высокой самооценки.

Типы людей

- Любой человек принадлежит к одному из трех типов:
- **Лидер**. Человек, который стремится управлять другими людьми, проектами, для которого нестерпимо быть просто ≪винтиком≫ в сложном механизме. Из лидеров выходят политические деятели, менеджеры разного уровня, для них, прежде всего, важен личный успех.
- **Технарь**. Человек, который получает настоящее удовольствие от самого процесса поиска решения, например, программирования, которому по большому счету наплевать, что есть начальники и подчиненные, что о нем не пишут газеты и т.д. Главное – решить задачу.
- **Общительный**. Эти люди разносят информацию, популяризуют результаты других людей. Психологи говорят, что 60% женщин относятся именно к этой категории.

Знание этих категорий очень важно для менеджеров программистских коллективов. Если собрать коллектив из одних лидеров, то будет постоянная борьба за власть, даже самые лучшие идеи не будут доведены до реализации, да и обмена идеями, скорее всего, не будет.

Команда из одних технарей не будет соблюдать бюджет и сроки, каждый будет сидеть в своем углу и решать ту задачу, которая ему больше нравится, а не ту, решение которой необходимо в данный момент Для мат-меха это очень типичная ситуация.

Наконец, в коллективе, состоящем преимущественно из общительных людей, будут самые веселые праздники, туда всегда будет приятно зайти, но работа спориться не будет.

Понятно, что хороший коллектив должен включать удачное сочетание лидеров, технарей и людей, ориентированных на общение. Как говорится, ≪мамы разные нужны, мамы всякие важны≫.

9. Организация коллектива разработчиков

- Проблема интерфейсов – главная проблема в организации коллектива. Если есть n сотрудников, то имеется Cn2 = n*(n-1)/2 парных интерфейсов, а ведь иногда надо обсудить проблему и втроем, и вчетвером.
- Таким образом, с ростом коллектива быстро растет количество интерфейсов внутри него, при каждом взаимодействии могут возникнуть споры, непонимание, разночтение и другие конфликты.
- Основная идея состоит в построении иерархии подчиненности. Есть руководитель темы или отдела, у него в подчинении несколько руководителей групп, у каждого руководителя группы в подчинении несколько специалистов. Разумеется, уровней иерархии может быть и 2, и 3, и 4, но все-таки не 15 и не 20. Каждый руководитель должен сформулировать задачу своим подчиненным так, чтобы минимизировать интерфейсы между ними, т.е. максимально локализовать решаемые ими задачи. Это не всегда возможно, не у всех руководителей получается, но так и различаются руководители и их команды.
- В незапамятные времена многие верили в закон Конвея: ≪Каждая система структурно подобна коллективу, ее разработавшему≫

Организация коллектива разработчиков

- Со временем закон Конвея, который был сформулирован его автором только в качестве шутки, был обобщен до довольно конструктивного метода – матричного. К чему подталкивает закон Конвея? Каждый специалист выполняет только свой небольшой кусок работы, но делает это всегда, для всех заказов такого же типа, набивает руку, почти не думает, совершает мало ошибок, т.е. процесс превращается в производственный.
- Итак, пусть у нас есть n специалистов и m однотипных заказов.
- Строим матрицу из **n** строк и **m** столбцов. Элемент в i-ой строке из j-ого столбца обозначает работу i-ого специалиста для j-ого заказа. Например, один специалист хорошо делает синтаксические анализаторы, другой – оптимизаторы, третий – генераторы и т.д., а однотипные заказы – это трансляторы с разных языков для разных платформ. Тем самым каждый специалист подчинен двум руководителям – административному (он же принадлежит какой-то группе, отделу) и руководителю проекта, в котором он принимает участие в данный момент.

Организация коллектива разработчиков

- У матричного метода есть свои плюсы и минусы.

С одной стороны, узкая специализация по конкретной теме позволяет надежнее планировать сроки, добиться той самой повторяемости результатов.

С другой стороны, нарушен принцип единоначалия (как когда-то в Красной Армии были командиры и комиссары, причем с одинаковыми правами и ответственностью; жизнь быстро доказала неправильность такого решения). Часто возникает ситуация, когда руководителю проекта нужен какой-то конкретный специалист, а начальник отдела загрузил его работой для другого заказа.

- Есть и еще один недостаток матричного метода, не столь очевидный. Конечно, занимаясь годами одним и тем же, специалист оттачивает свое мастерство, но в целом он ограничен, не расширяет свой кругозор и т.п. ≪Специалист подобен флюсу – полнота его односторонняя≫. Предположим, некий сотрудник института много лет успешно занимался темой X, а по прошествии какого-то времени тема X перестала быть актуальной.

- **И что он будет делать?**

Организация коллектива разработчиков

- У Брукса описана совершенно другая модель организации коллектива – бригада главного хирурга. Сложные операции всегда делает один человек – главный хирург, но ему помогает целая бригада – кто-то делает надрезы, а потом зашивает, кто-то подает инструменты, следит за показаниями приборов и т.д. Примерно такой же схемой воспользовались Миллз и его коллеги для разработки информационной системы газеты ≪Нью-Йорк Таймс≫.

- Все программы, документацию, основные тесты пишет один человек – главный программист. У него есть заместитель, который участвует в проектировании, обсуждениях, критикует решения главного программиста, но ни за что не отвечает. В случае болезни или по какой-то другой важной причине заместитель может заменить главного программиста.

- В бригаде есть тестер, секретарь, библиотекарь, продюсер с вполне понятными функциями. Возможно подключение еще каких-то узких специалистов. Оказалось, что такая бригада может работать в 3-5 раз быстрее традиционных команд программистов, поскольку не нужно тратить время на согласование деталей интерфейсов с другими программистами, изделие получается цельным и ≪элегантным≫, выполненным в одном стиле.

- Главный программист выбирается из числа наиболее опытных и высококвалифицированных, он может себе позволить сосредоточиться на основной задаче и не тратить время на всякие ≪бытовые≫ функции, которыми полна наша повседневная жизнь.

- Эта модель организации коллектива не нашла широкого применения. Во-первых, ее трудно масштабировать. Сколько строк исходного кода может написать один программист? Ну, 50 тысяч, ну, скажем, 100. А если нужно миллион строк? Тогда все проблемы интерфейсов возвращаются, да еще на более сложном уровне, поскольку двум главным программистам договориться гораздо труднее, чем двум рядовым. Во-вторых, где вы найдете множество программистов, готовых безропотно подчиняться главному программисту? Наша специальность подразумевает наличие твердого характера, творческого начала, гордости за свое детище.

10. Организация коллектива разработчиков в компании Microsoft

Служба Microsoft Consulting Services провела анализ результатов выполнения большого количества программных проектов. Оказалось, что только 24% проектов можно признать в той или иной степени успешными, 26% не были завершены, а остальные столкнулись с большими проблемами, например, бюджет был превышен вдвое или затрачено в 1,5 раза больше времени.

Основными причинами неудач были признаны следующие:

- постоянное изменение требований;
- нечеткие или неполные спецификации;
- низкое качество кода;
- слишком широкая постановка задачи;
- ошибка в подборе кадров;
- плохая организация работы;
- нечетко сформулированные цели.

Организация коллектива разработчиков в компании Microsoft

Для преодоления этих трудностей был предложен набор моделей Microsoft Solution Framework (MSF), в котором учтен опыт, накопленный группами разработки программных продуктов.

Самыми революционными оказались модель команды разработчиков и модель процесса разработки.

Первая модель (team model) описывает, как должны быть организованы коллективы и какими принципами им надо руководствоваться для достижения успеха в разработке программ.

Разные коллективы могут по-своему применять на практике различные элементы этой модели – все зависит от масштаба проекта, размера коллектива и квалификации его участников.

Формирование коллектива – сложная задача, которая должна решаться с помощью психологов. Вот некоторые основные положения:

- не должно быть команды из одних лидеров;
- не должно быть команды из одних исполнителей;
- в случае неудачи команда расформировывается;
- система штрафов (если проект проваливается – наказывают всех).

Этот ≪бублик≫ описывает только роли, за ними могут скрываться несколько человек, исполняющих каждую роль. Самое удивительное, что в этой модели не предусмотрено единоначалия – все роли важны, все роли равноправны, поэтому MSF называют моделью равных (team of peers).

Организация коллектива разработчиков в компании Microsoft

- **Program management** – управление программой. Исполнитель этой роли отвечает за организацию (но не руководит!): осуществляет ведение графика работ, утренние 15-минутные совещания, обеспечивает соответствие стандартам и спецификациям, фиксацию нарушений, написание технической документации.
- **Product management** – управление продуктом. Исполнители этой роли отвечают за общение с заказчиком, написание спецификации, разъяснение задач разработчикам.
- **Development** – наиболее традиционная роль – разработка и начальное тестирование продукта.
- **User education** – обучение пользователей. Написание пользовательской документации, обучающих курсов, повышение эффективности работы пользователей.
- **Logistic management** – установка, сопровождение и техническая поддержка продукта, а также материально-техническое обеспечение работы коллектива.
- **Testing** – тестирование. Выявление и устранение недоработок, исправление ошибок, другие функции QA.
- Все решения принимаются коллективно, разделяется и ответственность в случае провала проекта.

Организация коллектива разработчиков в компании Microsoft

В MSF утверждается, что такую модель можно масштабировать, разбивая систему по функциям. Лично у меня это утверждение (как и коллективная ответственность) вызывает большие сомнения.

Модель процесса определяет, когда и какие работы должны быть выполнены.

Перечислим основные принципы и практические приемы, лежащие в основе модели:

- итеративный подход (последовательный выпуск версий);
- подготовка четкой документации;
- учет неопределенности будущего;
- учет компромиссов;
- управление рисками;
- поддержание ответственного отношения коллектива к срокам выпуска продукта;
- разбиение крупных проектов на более мелкие управляемые части;
- ежедневная сборка проекта;
- постоянный анализ хода работ.

Организация коллектива разработчиков в компании Microsoft

Process model имеет три основные особенности:

- разбиение всего процесса на фазы;
- введение опорных точек;
- итеративность.

Весь процесс разбивается на четыре взаимосвязанных фазы. Прежде чем переходить к следующей фазе, на предыдущей должны быть получены определенные результаты.

В принципе, ничего нового тут нет, это известная спиральная модель, но необычно разбиение трудоемкости по фазам, связанное с тем, что и назначение каждой фазы весьма своеобразно.

Организация коллектива разработчиков в компании Microsoft

Envisioning – выработка единого понимания проекта всеми членами коллектива. Эта фаза заканчивается разработкой формализованного документа:

- *problem statement* — описание задачи объемом не более одной страницы;
- *vision statement* — от чего хотим уйти, чего хотим добиться;
- *solution concept* — что хотим внедрить и как;
- *user profiles* — кто будет этим пользоваться;
- *business goals* — возврат инвестиций;
- *design goals* — конкретные цели и ограничения продукта, его конкретные свойства.

Planning — планирование очередного цикла разработки:

- функциональные спецификации;
- план-график работ;
- оценка рисков.

Developing — разработка, причем рекомендуются различные технологические приемы, например, переиспользование кусков кода, программирование по контракту, написание защищенного от ошибок ПО и т.д.

Организация коллектива разработчиков в компании Microsoft

- Важную роль играют опорные точки (milestones), в которых анализируется состояние работ и производится их синхронизация. В этих точках приложение или его спецификации не замораживаются. Опорные точки позволяют проанализировать состояние проекта и внести необходимые коррективы, например, перестроиться под изменившиеся требования заказчика или отреагировать на риски, возможные в ходе дальнейшей работы. Для каждой опорной точки определяется, какие результаты должны быть получены к этому моменту.

- Каждая фаза процесса разработки завершается главной опорной точкой (major milestone). Характеризующие ее результаты видны не только коллективу разработчиков, но и заказчику. Главная опорная точка – это момент, когда все члены коллектива синхронизируют полученные результаты. Назначение таких точек в том, что они позволяют оценить жизнеспособность проекта. После анализа результатов коллектив разработчиков и заказчик совместно решают, можно ли переходить на следующую фазу. Таким образом, главные опорные точки – это критерии перехода с одной фазы проекта на другую.

- Внутри каждой фазы определяются промежуточные опорные точки (interium milestones). Они, как и главные, служат для анализа и синхронизации достигнутого, а не для замораживания проекта. Но, в отличие от главных опорных точек, промежуточные видны только членам коллектива разработчиков. Промежуточные опорные точки отмечают более скромные достижения и разбивают большую задачу на мелкие части, выполнение которых легче контролировать.

- Итеративность процесса заключается в его многократном повторении на протяжении всего цикла создания и существования продукта. На каждой успешной итерации в продукт включаются только те новые средства и функции, которые удовлетворяют изменяющимся требованиям бизнеса.

Важную роль в MSF играет postmortem – так раньше называли распечатки памяти после аварийных завершений программы. В этом документе описывается, что было хорошо и какие возникали проблемы, т.е. такие знания,

11. Документирование

- Одним из главных отличий просто программы от программного продукта является наличие разнообразной, хорошо подготовленной документации.
- Чтобы как-то структурировать этот раздел, воспользуемся ГОСТом ЕСПД (Единая Система Программной Документации) еще советских времен.

Техническое Задание

- Самым главным документом является ТЗ, в котором описываются цели и задачи работы, заказчик и исполнители, технические требования, сроки и этапы, требования секретности, форс-мажорные обстоятельства и правила предъявления результатов.
- Технические требования, в свою очередь, делятся на функциональные, экономические, требования по надежности, эффективности, защищенности от несанкционированного доступа и др.
- ТЗ должно быть составлено таким образом, чтобы исключить возможные разночтения, все требования должны быть сформулированы так, чтобы их можно было проверить однозначным образом.
- Самая главная ошибка начинающих руководителей – это неаккуратно составленное ТЗ или неоднозначные спецификации.
- Существует множество примеров, когда заказчики, пользуясь неопытностью наших руководителей, отказывались от финальных платежей, требовали дополнительных работ или еще каких-то преференций.

Документирование

- Следующим по важности документом является ПМИ (Программа и Методика Испытаний). Структурно ПМИ подобна ТЗ – практически для каждого пункта ТЗ в ПМИ говорится, как этот пункт будет проверяться. Способы проверки могут быть самыми разными – от пропуска специального теста до изучения исходных текстов программы, но они должны быть предусмотрены заранее, а не придумываться в момент испытаний.

- Новички приступают к составлению ПМИ непосредственно перед завершением работ, а опытные руководители составляют ПМИ практически одновременно с ТЗ (хотя бы в общих чертах) и согласовывают ее с заказчиками вместе с ТЗ. Именно хорошо составленная ПМИ является гарантией успешной сдачи работ.

- **Руководство системного программиста**, на современном чисто русском языке называется руководством по инсталляции. В нем описывается порядок установки системы на ЭВМ, как проверить корректность поставленной системы, как вносить изменения и т.п. Обычно это простой короткий документ; в противном случае система, наверное, плохая, сделанная непрофессионалами.

- **Руководство оператора (пользователя)** – это, собственно, основной документ, описывающий, как пользоваться системой. В хорошем руководстве сначала описывается идея системы, основные функции и как ими пользоваться, а уже потом идет описание всех клавиш и меню. Многие современные книги по программам Microsoft представляют собой яркие примеры никуда не годных руководств – можно прочесть 100 страниц и не понять основных функций.

Документирование

- **Руководство программиста.** Часто это самый объемный документ, описывающий внутреннюю организацию программы. Обычно этот документ идет в паре с документом ≪текст программы≫ – одностраничный документ с оглавлением CD. Руководство программиста дает заказчику возможность дописать какие-то новые фрагменты программы или переделать старые. В современной литературе этот документ называется SDK (Software Development Kit). Продукт, снабженный SDK, может стоить на порядок дороже, чем такой же продукт без оного, так что можно понять, насколько трудно создать действительно полезный SDK. Сейчас не очень принято продавать исходные тексты программ – проблемы с интеллектуальной собственностью, даже при наличии SDK трудно ≪влезть≫ в чужую программу – как говорится, себе дороже. Поэтому большое распространение получили API (Application Program Interface). Программа передается только в виде DLL (библиотека двоичных кодов), но известно, как обратиться к каждой функции из других программ, т.е. известно имя точки входа, количество, типы и значения параметров. Наличие множества API, конечно, хуже, чем наличие исходных текстов (например, нельзя переделать что-то в середине функции), зато много проще в использовании. С другой стороны, все большую популярность приобретает FSF (Free Software Foundation). Основателем этого движения был Ричард Столман, который забил тревогу по поводу попыток крупных фирм запатентовать многие основные алгоритмы и программы: ≪Дойдет до того, что они запатентуют понятия ≪цикл≫ и ≪подпрограмма≫, что мы будем тогда делать?≫ FSF представляет собой собрание программ в исходных текстах; любой программист может свободно использовать их в своих целях, но все добавления и улучшения, которые он сделал, тоже следует положить в FSF. Таким образом, FSF представляет собой одно из самых больших доступных хранилищ программ. Многие ведомства (например, военные) просто не могут использовать программы, текстов которых они не имеют. Мы уже много раз пользовались FSF, естественно, соблюдая все правила игры.

- Остальные документы, перечисленные в ЕСПД, носят формальный или необязательный характер, поэтому мы их обсуждать не будем.