Лекция 1. Понятие качества программного обеспечения.

1. Качество как основная характеристика программного обеспечения

Разработка ПС достигла такого уровня развития, что стало необходимо использовать инженерные методы, в том числе для оценивания результатов проектирования на этапах ЖЦ, контроля достижения показателей качества и метрического их анализа, оценки риска и степени использования готовых компонентов для снижения стоимости разработки нового проекта. Основу инженерных методов в программировании составляет повышение качества, для достижения которого сформировались методы определения требований к качеству, подходы к выбору и усовершенствованию моделей метрического анализа показателей качества, методы количественного измерения показателей качества на этапах ЖЦ.

Качество программного обеспечения – это предмет стандартизации. Стандарт ГОСТ 2844-94 дает определение **качества ПО** как совокупность свойств (показателей качества) ПО, которые обеспечивают его способность удовлетворять потребности заказчика в соответствии с назначением. Стандарт ISO/IEC 25000:2014 даёт следующее определение понятия **качество программного обеспечения** – способность программного продукта при заданных условиях удовлетворять установленным или предполагаемым потребностям.

Согласно стандарту на этапах ЖЦ должен проводиться контроль качества ПО:

– проверка соответствия требований проектируемому продукту и критериев их достижения;

– верификация и аттестация (валидация) промежуточных результатов ПО на этапах ЖЦ и измерение степени удовлетворения достигаемых отдельных показателей;тестирование готовой ПС, сбор данных об отказах, дефектах и других ошибках, обнаруженных в системе;

– подбор моделей надежности для оценивания надежности по полученным результатам тестирования (дефекты, отказы и др.);

– оценка показателей качества, заданных в требованиях на разработку ПС.

Качество ПО - это относительное понятие, которое имеет смысл только при учете реальных условий его применения, поэтому требования, предъявляемые к качеству, ставятся в соответствии с условиями и конкретной областью их применения. Оно характеризуется тремя аспектами: качество программного продукта, качество процессов ЖЦ и качество сопровождения или внедрения.

Аспект, связанный с процессами ЖЦ, определяет степень формализации, достоверности самих процессов ЖЦ разработки ПО, а также верификацию и валидацию промежуточных результатов на этих процессах. Поиск и устранение ошибок в готовом ПО

проводится методами тестирования, которые снижают количество ошибок и повышают качество этого продукта.

Качество продукта достигается процедурами контроля промежуточных продуктов на процессах ЖЦ, проверкой их на достижение необходимого качества, а также методами сопровождения продукта. Эффект от внедрения ПС в значительной степени зависит от знаний обслуживающего персонала функций продукта и правил их выполнения.Модель качества ПО имеет следующие четыре уровня представления.

Первый уровень соответствует определению характеристик (показателей) качества ПО, каждая из которых отражает отдельную точку зрения пользователя на качество. Согласно стандарту [10.1-10.4] в модель качества входит шесть характеристик или шесть показателей качества:

1. функциональность (functionality);
2. надежность (realibility);
3. удобство (usability);
4. эффективность (efficiency);
5. сопровождаемость (maitainnability);
6. переносимость (portability).

Второму уровню соответствуют атрибуты для каждой характеристики качества, которые детализируют разные аспекты конкретной характеристики. Набор атрибутов характеристик качества используется при оценке качества.

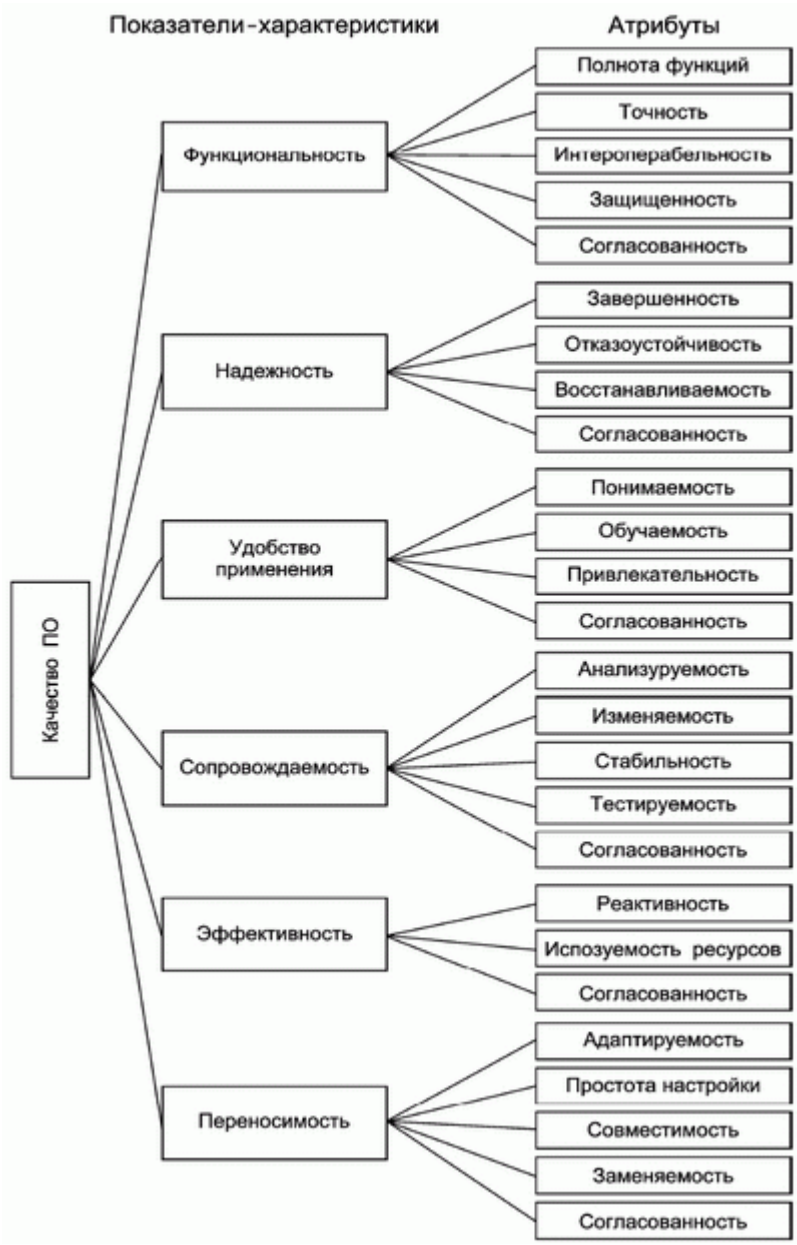
Третий уровень предназначен для измерения качества с помощью метрик, каждая из них согласно стандарту [10.1] определяется как комбинация метода измерения атрибута и шкалы измерения значений атрибутов. Для оценки атрибутов качества на этапах ЖЦ (при просмотре документации, программ и результатов тестирования программ) используются метрики с заданным оценочным весом для нивелирования результатов метрического анализа совокупных атрибутов конкретного показателя и качества в целом. Атрибут качества определяется с помощью одной или нескольких методик оценки на этапах ЖЦ и на завершающем этапе разработки ПО.

Четвертый уровень - это оценочный элемент метрики (вес), который используется для оценки количественного или качественного значения отдельного атрибута показателя ПО. В зависимости от назначения, особенностей и условий сопровождения ПО выбираются наиболее важные характеристики качества и их атрибуты (рис. 10.2).

Выбранные атрибуты и их приоритеты отражаются в требованиях на разработку систем либо используется соответствующие приоритеты эталона класса ПО, к которому это ПО относится.

2. Показатели качества программного обеспечения.

Выделяют следующие показатели качества программного обеспечения:



1. **Функциональность**. Группа свойств ПО, обуславливающая его способность выполнять определенный перечень функций, которые удовлетворяют потребностям в соответствии с назначением Под функцией понимается некоторая упорядоченная последовательность действий для удовлетворения потребительских свойств. Функции бывают целевые (основные) и вспомогательные. К атрибутам функциональности относятся:

– ***функциональная полнота*** - свойство компонента, которое показывает степень достаточности основных функций для решения задач в соответствии с назначением ПО;

– ***правильность (точность)*** - атрибут, который показывает степень достижения правильных результатов;

– ***интероперабельность*** - атрибут, который показывает возможность взаимодействовать на ПО специальными системами и средами (ОС, сеть);

– ***защищенность*** - атрибут, который показывает на способность ПО предотвращать несанкционированный доступ (случайный или умышленный) к программам и данным.

2. **Надёжность**. Группа свойств, обусловливающая способность ПО сохранять работоспособность и преобразовывать исходные данные в результат за установленный период времени, характер отказов которого является следствием внутренних дефектов и условий его применения. Снижение надежности ПО происходит из-за ошибок в требованиях, проектировании и выполнении. Отказы и ошибки в программах появляются на заданном промежутке времени.

К подхарактеристикам надежности ПО относятся:

– ***безотказность*** - атрибут, который определяет способность ПО функционировать без отказов (как программы, так и оборудования);

– ***устойчивость*** к ошибкам - атрибут, который показывает на способность ПО выполнять функции при аномальных условиях (сбой аппаратуры, ошибки в данных и интерфейсах, нарушение в действиях оператора и др.);

– ***восстанавливаемость*** - атрибут, который показывает на способность программы к перезапуску для повторного выполнения и восстановления данных после отказов.

К некоторым типам систем (реального времени, радарных, систем безопасности, коммуникация и др.) предъявляются требования для обеспечения высокой надежности (недопустимость ошибок, точность, достоверность, удобство применения и др.). Таким образом, надежность ПО в значительной степени зависит от числа оставшихся и не устраненных ошибок в процессе разработки на этапах ЖЦ. В ходе эксплуатации ошибки обнаруживаются и устраняются.

Если при исправлении ошибок не вносятся новые или, по крайней мере, новых ошибок вносится меньше, чем устраняется, то в ходе эксплуатации надежность ПО непрерывно возрастает. Чем интенсивнее проводится эксплуатация, тем интенсивнее выявляются ошибки и быстрее растет надежность ПО.

К факторам, влияющим на надежность ПО, относятся:

– совокупность угроз, приводящих к неблагоприятным последствиям и к ущербу системы или среды ее функционирования;

– угроза как проявление нарушения безопасности системы;

– целостность как способность системы сохранять устойчивость работы и не иметь риска.

Обнаруженные ошибки могут быть результатом угрозы извне или отказов, они повышают риск и уменьшают некоторые свойства надежности системы.

Надежность - одна из ключевых проблем современных программных систем, и ее роль будет постоянно возрастать, поскольку постоянно повышаются требования к качеству компьютерных систем. Новое направление - инженерия программной надежности (*Software reliability* engineering) - ориентировано на количественное изучение операционного поведения компонентов системы по отношению к пользователю, ожидающему надежную работу системы [10.7], и включает:

– измерение надежности, т.е. проведение ее количественной оценки с помощью предсказаний, сбора данных о поведении системы в процессе эксплуатации и современных моделей надежности;

– стратегии и метрики конструирования и выбора готовых компонентов, процесс разработки компонентной системы, а также среда функционирования, влияющая на надежность работы системы;

– применение современных методов инспектирования, верификации, валидации и тестирования при разработке систем, а также при эксплуатации.

Верификация применяется для определения соответствия готового ПО установленным спецификациям, а валидация - для установления соответствия системы требованиям пользователя, которые были предъявлены заказчиком.

В инженерии надежности термин dependability (пригодноспособность) обозначает способность системы иметь свойства, желательные для пользователя, который уверен в качественном выполнении функций ПС, заданных в требованиях. Данный термин определяется дополнительным количеством атрибутов, которыми должна обладать система, а именно:

– готовность к использованию (availability);

– готовностью к непрерывному функционированию (reliability);

– безопасность для окружающей среды, т.е. способность системы не вызывать катастрофических последствий в случае отказа (safety);

– секретность и сохранность информации (confidential);

– способность к сохранению системы и устойчивости к самопроизвольному ее изменению (integrity);

– способность к эксплуатации ПО, простота выполнения операций обслуживания, а также устранения ошибок, восстановление системы после их устранения и т.п. (maintainability);

– готовность и сохранность информации (security) и др.

Достижение надежности системы обеспечивается предотвращением отказа (fault prevention), его устранением (removal fault), а также оценкой возможности появления новых отказов и мер борьбы с ними с применением методов теории вероятности.

Каждый программный компонент, его операции и данные обрабатываются в дискретные моменты времени, например, $\delta, 2\delta, \ldots n\delta$. Пусть за время T после первого неудачно обработанного компонента системы появился отказ, q_b - вероятность этой неудачи, тогда $P\{T > n\delta\} = (1 - q_s)^n$ и среднее время ожидания $T = \delta/q_s$.

Положим, что момент времени δ убывает, а время T остается фиксированным, тогда имеем $P\{T > t\} = (1 - \frac{\delta}{T})\frac{t}{\delta} = e^{-\frac{t}{T}}$, время до отказа в данном случае - непрерывная величина $c \ldots l$, распределенная экспоненциально с параметром $T, \ldots, 1/T$.

Таким образом, оценка надежности ПО - это трудоемкий процесс, требующий создания устойчивой работы системы по отношению к отказам ПО, т.е. вероятности того, что система восстановится самопроизвольно в некоторой точке после возникновения в ней отказа (fault).

3. **Удобство применения.** Совокупность свойств ПО для предполагаемого круга пользователей и отражающих легкость его освоения и адаптации к изменяющимся

условиям эксплуатации, стабильность работы и подготовки данных, понимаемость результатов, удобства внесения изменений в программную документацию и в программы. В стандарте [10.3] удобство применения определено как специфическое множество атрибутов программного продукта, характеризующих его эргономичность.

К подхарактеристиками удобства применения относятся:

- ***понимаемость*** - атрибут, который определяет усилия, затрачиваемые на распознавание логических концепций и условий применения ПО;
- ***изучаемость*** (легкость изучения) - атрибут, который определяет усилия пользователей на определение применимости ПО путем использования операционного контроля, диагностики, а также процедур, правил и документации;
- ***оперативность*** - атрибут, который показывает на реакцию системы при выполнении операций и операционного контроля;
- ***согласованность*** - атрибут, который показывает соответствие разработки требованиям стандартов, соглашений, правил, законов и предписаний.

4. **Эффективность** - множество атрибутов, которые определяют взаимосвязь уровней выполнения ПО, использования ресурсов (средства, аппаратура, материалы - бумага для печатающего устройства и др.) и услуг, выполняемых штатным обслуживающим персоналом и др. К подхарактеристикам эффективности ПО относятся:

- ***реактивность*** - атрибут, который показывает время отклика, обработки и выполнения функций;
- ***эффективность ресурсов*** - атрибут, показывающий количество и продолжительность используемых ресурсов при выполнении функций ПО;
- ***согласованность*** - атрибут, который показывает соответствие данного атрибута с заданными стандартами, правилами и предписаниями.

5. **Сопровождаемость** – множество свойств, которые показывают на усилия, которые надо затратить на проведение модификаций, включающих корректировку, усовершенствование и адаптацию ПО при изменении среды, требований или функциональных спецификаций. Сопровождаемость включает подхарактеристики:

- ***анализируемость*** - атрибут, определяющий необходимые усилия для диагностики отказов или идентификации частей, которые будут модифицироваться;изменяемость - атрибут, который определяет удаление ошибок в ПО или внесение изменений для их устранения, а также введение новых возможностей в ПО или в среду функционирования;
- ***стабильность*** - атрибут, указывающий на постоянство структуры и риск ее модификации;
- ***тестируемость*** - атрибут, показывающий на усилия при проведении валидации и верификации с целью обнаружения несоответствий требованиям, а также на необходимость проведения модификации ПО и сертификации;
- ***согласованность*** - атрибут, который показывает соответствие данного атрибута соглашениям, правилам и предписаниям стандарта.

6. **Переносимость** - множество показателей, указывающих на способность ПО адаптироваться к работе в новых условиях среды выполнения. Среда может быть организационной, аппаратной и программной. Поэтому перенос ПО в новую среду выполнения может быть связан с совокупностью действий, направленных на обеспечение

его функционирования в среде, отличной от той среды, в которой оно создавалось с учетом новых программных, организационных и технических возможностей. Переносимость включает подхарактеристики:

– ***адаптивность*** - атрибут, определяющий усилия, затрачиваемые на адаптацию к различным средам;

– ***настраиваемость*** (простота инсталляции) - атрибут, который определяет необходимые усилия для запуска данного ПО в специальной среде;

– ***сосуществование*** - атрибут, который определяет возможность использования специального ПО в среде действующей системы;

– ***заменяемость*** - атрибут, который обеспечивают возможность интероперабельности при совместной работе с другими программами с необходимой инсталляцией или адаптацией ПО;

– ***согласованность*** - атрибут, который показывает на соответствие стандартам или соглашениями по обеспечению переноса ПО.

7. **Совместимость**: пригодность продукции, процессов или услуг к совместному, но не вызывающему нежелательных взаимодействий использованию при заданных условиях для выполнения установленных требований

3. Метрики качества программного обеспечения.

В настоящее время в программной инженерии еще не сформировалась окончательно система метрик. Действуют разные подходы к определению их набора и методов измерения.

Система измерения включает метрики и модели измерений, которые используются для количественной оценки качества ПО.

При определении требований к ПО задаются соответствующие им внешние характеристики и их атрибуты (подхарактеристики), определяющие разные стороны управления продуктом в заданной среде. Для набора характеристик качества ПО, приведенных в требованиях, определяются соответствующие метрики, модели их оценки и диапазон значений мер для измерения отдельных атрибутов качества.

Согласно стандарту метрики определяются по модели измерения атрибутов ПО на всех этапах ЖЦ (промежуточная, внутренняя метрика) и особенно на этапе тестирования или функционирования (внешние метрики) продукта.

Остановимся на классификации метрик ПО, правилах для проведения метрического анализа и процесса их измерения.

Существует три типа метрик:

- **метрики программного продукта**, которые используются при измерении его характеристик - свойств;
- **метрики процесса**, которые используются при измерении свойства процесса ЖЦ создания продукта.
- метрики использования.

Метрики программного продукта включают:

- внешние метрики, обозначающие свойства продукта, видимые пользователю;
- внутренние метрики, обозначающие свойства, видимые только команде разработчиков.

Внешние метрики продукта - это метрики:

- надежности продукта, которые служат для определения числа дефектов;
- функциональности, с помощью которых устанавливаются наличие и правильность реализации функций в продукте;
- сопровождения, с помощью которых измеряются ресурсы продукта (скорость, память, среда);применимости продукта, которые способствуют определению степени доступности для изучения и использования;
- стоимости, которыми определяется стоимость созданного продукта.

Внутренние метрики продукта включают:

- метрики размера, необходимые для измерения продукта с помощью его внутренних характеристик;
- метрики сложности, необходимые для определения сложности продукта;
- метрики стиля, которые служат для определения подходов и технологий создания отдельных компонентов продукта и его документов.

Внутренние метрики позволяют определить производительность продукта и являются релевантными по отношению к внешним метрикам.

Внешние и внутренние метрики задаются на этапе формирования требований к ПО и являются предметом планирования и управления достижением качества конечного программного продукта.

Метрики продукта часто описываются комплексом моделей для установки различных свойств, значений модели качества или прогнозирования. Измерения проводятся, как правило, после калибровки метрик на ранних этапах проекта. Общая мера - степень трассируемости, которая определяется числом трасс, прослеживаемых с помощью моделей сценариев типа UML и оценкой количества:

- требований;
- сценариев и действующих лиц;
- объектов, включенных в сценарий, и локализация требований к каждому сценарию;
- параметров и операций объекта и др.

Стандарт ISO/IEC 9126-2 определяет следующие типы мер:

- мера размера ПО в разных единицах измерения (число функций, строк в программе, размер дисковой памяти и др.);
- мера времени (функционирования системы, выполнения компонента и др.);
- мера усилий (производительность труда, трудоемкость и др.);
- мера учета (количество ошибок, число отказов, ответов системы и др.).

Специальной мерой может служить уровень использования повторных компонентов и измеряется как отношение размера продукта, изготовленного из готовых компонентов, к размеру системы в целом. Данная мера используется также при определении стоимости и качества ПО. Примеры метрик:

- общее число объектов и число повторно используемых;
- общее число операций, повторно используемых и новых операций;
- число классов, наследующих специфические операции;
- число классов, от которых зависит данный класс;
- число пользователей класса или операций и др.

При оценке общего количества некоторых величин часто используются среднестатистические метрики (среднее число операций в классе, наследников класса или операций класса и др.).

Как правило, меры в значительной степени являются субъективными и зависят от знаний экспертов, производящих количественные оценки атрибутов компонентов программного продукта.

Примером широко используемых внешних метрик программ являются метрики Холстеда - это характеристики программ, выявляемые на основе статической структуры программы на конкретном языке программирования: число вхождений наиболее часто встречающихся операндов и операторов; длина описания программы как сумма числа вхождений всех операндов и операторов и др.

Качество ПО - это совокупность свойств, определяющих полезность изделия (программы) для пользователей в соответствии с функциональным назначением и предъявлёнными требованиями.

Характеристика качества программы - понятие, отражающее отдельные факторы, влияющие на качество программ и поддающиеся измерению.

Критерий качества - численный показатель, характеризующий степень, в которой программе присуще оцениваемое свойство.

Критерии качества включают следующие характеристики : экономичность, документированность, гибкость, модульность, надёжность, обоснованность, тестируемость, ясность, точность, модифицируемость, эффективность, легкость сопровождения и т.д.

Критерий должен :

* численно характеризовать основную целевую функцию программы;

* обеспечивать возможность определения затрат, необходимых для достижения требуемого уровня качества, а также степени влияния на показатель качества различных внешних факторов;

* быть по возможности простым, хорошо измеримым и иметь малую дисперсию.

Для измерения характеристик и критериев качества используют метрики.

Метрики Холстеда

Метрики Холстеда включают следующие характеристики.

1) Длина программы:

N'= n1 x log2(n1) + n2 x log2(n2)

Объем программы:

V = N * log2(n) (бит)

2) Потенциальный (минимальный) объем:

V* = (2+ n2*) * log2(2+ n2*)

3) Граничный объем:

V** = (2+ (n2*)* log2(n2*)) * log2(2+ n2*)

5) Соотношения между операциями и операндами (зависимость числа операндов n2 от числа операций n1):

A = n2* /(n2* +2) * log2(n2*/2)

B = n2* - 2 * A

n2 = A * n1 + B

6) Уровень программы:

L = V*/V

где V* - потенциальный объем, V - объем программы.

Альтернативное определение уровня L:

L' = (n1*) * n2/(n1*N2)

где n1*=2

Альтернативным методом определение уровня программы получили значение на 0.001 превышающее значение полученное в первом методе.

7) Интеллектуальное содержание:

I = L' * V

8) Работа по программированию (общее число элементарных мысленных различий, требуемых для порождения программы):

E = V 2/ V*

9) Приближенное время программирования:

T' = E/S

где S = 18 моментов (различий)/сек - постоянная Страуда.

T' =6882(сек)

Время программирования указанное в техническом задании – 1 час

10) Уровень языка:

A = L * L* V

Уровень языка определяет его производительность.

11) Уравнение ошибок:

Число переданных ошибок в программе: B = V/E0

где E0 = V* x V* x V*/(A x A) - среднее число элементарных различений между возможными ошибками в программировании.

Исходя из полученных данных, можно предположить, что качество реализации алгоритма в большей степени определяется следующими параметрами:

* интеллектуальное содержание
* работа по программированию
* приближенное время программирования
* уровень языка
* уравнение ошибок

Описание метрик Холстеда

Метрики Холстеда предлагают разумный подход к решению следующих задач:

- предсказание условий, необходимых для программирования по предложенным проектам;
- определение норм первоначальных ошибок;
- количественная оценка языков программирования и эффекта модульности;
- обоснование метода измерения различий между программами, написанными специалистами разного уровня.

В основе вычисления метрик Холстеда лежит концепция, согласно которой алгоритм состоит только из операторов и операндов (проверяется рассмотрением простых вычислительных машин с форматом команд, содержащим две части: код операции и адрес операнда). Операнды - переменные или константы, используемые в данной реализации алгоритма. Операторы - комбинации символов, влияющие на значение или порядок операндов.

В основе вычисляемых свойств алгоритма лежат следующие характеристики:

- n1 - число различных операторов данной реализации;
- n2 - число различных операндов данной реализации;
- N1 - общее число всех операторов;
- N2 - общее число всех операндов;

- n2*- число различных входных и выходных операндов

На основании приведенных выше характеристик вычисляются:

словарь n = n1 + n2;

длина реализации N = N1 + N2.

объем программы V=N*log2(n) (бит).

Под битом подразумевается логическая единица информации - символ, оператор, операнд.

Далее М. Холстед вводит n* - теоретический словарь программы, т.е. словарный запас, необходимый для написания программы, с учетом того, что необходимая функция уже реализована в данном языке и, следовательно, программа сводится к вызову этой функции. Например, согласно М. Холстеду, возможное осуществление процедуры выделения простого числа могло бы выглядеть так:

- CALL SIMPLE (X,Y),
- где Y - массив численных значений, содержащий искомое число X.
- Теоретический словарь в этом случае будет состоять из
- n1* : {CALL, SIMPLE (...)},
- n1*=2; n2* : {X, Y},
- n2*=2,
- а его длина, определяемая как
- n* = n1* + n2*,
- будет равняться 4.
- Используя n*, Холстед вводит оценку V*:
- V* = n* * log2 n*, (3)

с помощью которой описывается потенциальный объем программы, соответствующий максимально компактному тексту программы, реализующей данный алгоритм.

На основе этих атрибутов можно вычислить время программирования, уровень программы (структурированность и качество) и языка программирования (абстракции средств языка и ориентация на проблему) и др.

В качестве метрик процесса могут быть время разработки, число ошибок, найденных на этапе тестирования и др. Практически используются следующие метрики процесса:

- общее время разработки и отдельно время для каждой стадии;
- время модификации моделей;
- время выполнения работ на процессе;
- число найденных ошибок при инспектировании;
- стоимость проверки качества;
- стоимость процесса разработки.

Метрики использования служат для измерения степени удовлетворения потребностей пользователя при решении его задач. Они помогают оценить не свойства самой программы, а результаты ее эксплуатации - эксплуатационное качество. Примером может служить - точность и полнота реализации задач пользователя, а такжезатраченные ресурсы (трудозатраты, производительность и др.) на эффективное решение задач пользователя. Оценка требований пользователя проводится с помощью внешних метрик.

4. Управление качеством программного обеспечения.

Под управлением качества понимается совокупность организационной структуры и ответственных лиц, а также процедур, процессов и ресурсов для планирования и управления достижением качества ПС. Управление качеством - SQM (Software Quality Management) базируется на применении стандартных положений по гарантии качества - SQA (Software Quality Assurance).

Цель процесса SQA состоит в гарантировании того, что продукты и процессы согласуются с требованиями, соответствуют планам и включают следующие виды деятельности:

- внедрение стандартов и соответствующих процедур разработки ПС на этапах ЖЦ;
- оценка соблюдения положений этих стандартов и процедур. Гарантия качества состоит в следующем:
- проверка непротиворечивости и выполнимости планов;
- согласование промежуточных рабочих продуктов с плановыми показателями;
- проверка изготовленных продуктов заданным требованиям;
- анализ применяемых процессов на соответствие договору и планам;
- согласование с заказчиком среды и методов разработки продукта;
- проверка принятых метрик продуктов, процессов и приемов их измерения в соответствии с утвержденным стандартом и процедурами измерения.

Цель процесса управления SQM - мониторинг (систематический контроль) качества для гарантии того, что продукт будет удовлетворять потребителю и предполагает выполнение следующего:

- определение количественных свойств качества, основанных на выявленных и предусмотренных потребностях пользователей;
- управление реализацией поставленных целей для достижения качества.

Процесс SQM основывается на гарантии того, что:

- цели достижения требуемого качества установлены для всех рабочих продуктов в контрольных точках продукта;
- определена стратегия достижения качества, метрики, критерии, приемы, требования к процессу измерения и др.;
- определены и выполняются действия, связанные с предоставлением продуктам свойств качества;
- проводится контроль качества (SQA, верификация и валидация) и целей;
- выполняются процессы измерения и оценивания конечного продукта на достижение требуемого качества.

Основные стандартные положения [10.1[10.2-10.4, 10.15] по созданию качественного продукта и оценки достигнутого его уровня позволяют выделить два процесса обеспечения качества на этапах ЖЦ:

- гарантия (подтверждение) качества ПС как результат определенной деятельности на каждом этапе ЖЦ с проверкой соответствия системы стандартам и процедурам, ориентированным на достижении качества;
- инженерия качества как процесс предоставления продуктам ПО свойств функциональности, надежности, сопровождения и других характеристик качества.

Процессы достижения качества предназначены для:

- управления, разработки и обеспечения гарантий в соответствии с указанными стандартами и процедурами;

• управления конфигурацией (идентификация, учет состояния и действий по аутентификации), риском и проектом в соответствии со стандартами и процедурами;

• контроль базовой версии ПС и реализованных в ней характеристик качества.

Выполнение указанных процессов включает такие действия:

• оценка стандартов и процедур, которые выполняются при разработке программ;

• ревизия управления, разработки и обеспечение гарантии качества ПО, а также проектной документации (отчеты, графики разработки, сообщения и др.);

• контроль проведения формальных инспекций и просмотров;

• анализ и контроль проведения приемочного тестирования (испытания) ПС.

Для организации, которая занимается разработкой ПС, в том числе из компонентов, инженерия качества ПС должна поддерживаться системой управлением качеством (планирование, учет и контроль).

Инженерия качества включает набор методов и мероприятий, с помощью которых программные продукты проверяются на выполнение требований к качеству и снабжаются характеристиками, предусмотренными в требованиях на ПО.

Система качества (Quality systems - QS) [10.15] - это набор организационных структур, методик, мероприятий, процессов и ресурсов для осуществления управления качеством. Для обеспечения требуемого уровня качества ПО применяются два подхода. Один из них ориентирован на конечный программный продукт, а второй - на процесс создания продукта.

При подходе, ориентированном на продукт, оценка качества проводится после испытания ПС. Этот подход базируется на предположении, что чем больше обнаружено и устранено ошибок в продукте при испытаниях, тем выше его качество.

При втором подходе предусматриваются и принимаются меры по предотвращению, оперативному выявлению и устранению ошибок, начиная с начальных этапов ЖЦ в соответствии с планом и процедурами обеспечения качества разрабатываемой ПС. Этот подход представлен в серии стандартов ISO 9000 и 9000-1,2,3, который дает рекомендации организациямразработчикам создавать систему качества согласно схемы, приведенной на рис. 10.3.

Важное место в инженерии качества отводится процессу измерения характеристик процессов ЖЦ, его ресурсов и создаваемых на них рабочих продуктов. Этот процесс реализуется группой качества,верификации и тестирования. В ее функции входит планирование, оперативное управление и обеспечение качества.

Планирование качества представляет собою деятельность, направленную на определение целей и требований к качеству. Оно охватывает идентификацию, установление целей, требований к качеству, классификацию и оценку качества. Составляется календарный планграфик для проведения анализа состояния разработки и последовательного измерения спланированных показателей и критериев на этапах ЖЦ.

Оперативное управление включает методы и виды деятельности оперативного характера для текущего управления процессом проектирования и устранения причин плохого или неудовлетворительного функционирования ПС.

Обеспечение качества заключается в выполнении и проверки того, что объект разработки выполняет указанные требования к качеству. Цели обеспечения качества могут быть внутренние и внешние. Внутренние цели - создание уверенности у руководителя проекта, что качество обеспечивается. Внешние цели - это создание

уверенности у пользователя, что требуемое качество достигнуто и получено качественное программное обеспечение.

Как показывает опыт, ряд фирм, выпускающих программную продукцию, имеют системы качества, что обеспечивает им производство конкурентоспособной продукции. Система качества включает мониторинг спроса на выпускаемый новый вид продукции, контроль всех звеньев его производства, включая подбор и поставку готовых компонентов для системы.

При отсутствии соответствующих служб качества разработчики ПО должны применять собственные нормативные и методические документы, регламентирующие процесс управления качеством ПО для всех категорий разработчиков и пользователей программной продукции.

5. Методы обеспечения критериев качества программного обеспечения

Ниже обсуждаются обеспечение примитивов качества ПС, выражающих критерии функциональности и надежности ПС.

11.2. Обеспечение завершенности программного средства

Завершенность ПС является общим примитивом качества ПС для выражения и функциональности и надежности ПС, причем для функциональности она является единственным примитивом (см. лекцию 4).

Функциональность ПС определяется его функциональной спецификацией. Завершенность ПС как примитив его качества является мерой того, в какой степени эта

спецификация реализована в разрабатываемом ПС. Обеспечение этого примитива в полном объеме означает реализацию каждой из функций, определенной в функциональной спецификации, со всеми указанными там деталями и особенностями. Все рассмотренные ранее технологические процессы показывают, как это может быть сделано.

Однако в спецификации качества ПС могут быть определены несколько уровней реализации функциональности ПС: может быть определена некоторая упрощенная (начальная или стартовая) версия, которая должна быть реализована в первую очередь; могут быть также определены и несколько промежуточных версий. В этом случае возникает дополнительная технологическая задача: организация наращивания функциональности ПС. Здесь важно отметить, что разработка упрощенной версии ПС не есть разработка его *прототипа*. Прототип разрабатывается для того, чтобы лучше понять условия применения будущего ПС [11.1], уточнить его внешнее описание. Он рассчитан на избранных пользователей и поэтому может сильно отличаться от требуемого ПС не только выполняемыми функциями, но и особенностями пользовательского интерфейса. Упрощенная же версия разрабатываемого ПС должна быть рассчитана на *практически полезное* применение любыми пользователями, для которых предназначено это ПС. Поэтому главный принцип обеспечения функциональности такого ПС заключается в том, чтобы с самого начала разрабатывать ПС таким образом, как будто требуется ПС в полном объеме, до тех пор, когда разработчики будут иметь дело непосредственно с теми частями или деталями ПС, реализацию которых можно отложить в соответствии со спецификацией его качества. Тем самым, и внешнее описание и описание архитектуры ПС должно быть разработано в полном объеме. Можно отложить лишь реализацию тех программных подсистем (определенных в архитектуре разрабатываемого ПС), функционирования которых не требуется в начальной версии этого ПС. Реализацию же самих программных подсистем лучше всего производить методом целенаправленной конструктивной реализации, оставляя в начальной версии ПС подходящие имитаторы тех программных модулей, функционирование которых в этой версии не требуется. Допустима также упрощенная реализация некоторых программных модулей, опускающая реализацию некоторых деталей соответствующих функций. Однако такие модули с технологической точки зрения лучше рассматривать как своеобразные их имитаторы (хотя и далеко продвинутые).

Достигнутый при обеспечении функциональности ПС уровень его завершенности на самом деле может быть не таким, как ожидалось, из-за ошибок, оставшихся в этом ПС. Можно лишь говорить, что требуемая завершенность достигнута с некоторой вероятностью, определяемой объемом и качеством проведенного тестирования. Для того чтобы повысить эту вероятность, необходимо продолжить тестирование и отладку ПС. Однако, оценивание такой вероятности является весьма специфической задачей, которая пока еще ждет соответствующих теоретических исследований.

11.3. Обеспечение точности программного средства

Обеспечение этого примитива качества связано с действиями над значениями вещественных типов (точнее говоря, со значениями, представляемыми с некоторой погрешностью). Обеспечить требуемую точность при вычислении значения той или иной функции - значит получить это значение с погрешностью, не выходящей за рамки заданных границ. Видами погрешности, методами их оценки и методами достижения требуемой точности (т.н. *приближенными вычислениями*) занимается вычислительная математика [11.1, 11.2]. Здесь мы лишь обратим внимание на некоторую структуру погрешности: погрешность вычисленного значения (*полная погрешность*) зависит

- от *погрешности используемого метода* вычисления (в которую мы включаем и неточность используемой модели),
- от погрешности представления используемых данных (от т.н. *неустранимой погрешности*),

- от *погрешности округления* (неточности выполнения используемых в методе операций).

11.4. Обеспечение автономности программного средства

Вопрос об автономности программного средства решается путем принятия решения о возможности использования в разрабатываемом ПС какого-либо подходящего базового программного обеспечения. Надежность имеющегося в распоряжении разработчиков базового программного обеспечения для целевого компьютера может не отвечать требованиям к надежности разрабатываемого ПС. Поэтому от использования такого программного обеспечения приходиться отказываться, а его функции в требуемом объеме приходится реализовывать в рамках разрабатываемого ПС. Аналогичное решение приходится принимать при жестких ограничениях на используемые ресурсы (по критерию эффективности ПС). Такое решение может быть принято уже в процессе разработки спецификации качества ПС, иногда – на этапе конструирования ПС.

11.5. Обеспечение устойчивости программного средства

Этот примитив качества ПС обеспечивается с помощью так называемого *защитного программирования*. Вообще говоря, защитное программирование применяется при программировании модуля для повышения надежности ПС в более широком смысле. Как утверждает Майерс [11.3], “защитное программирование основано на важной предпосылке: худшее, что может сделать модуль, – это принять неправильные входные данные и затем вернуть неверный, но правдоподобный результат”. Для того, чтобы этого избежать, в текст модуля включают проверки его входных и выходных данных на их корректность в соответствии со спецификацией этого модуля, в частности, должны быть проверены выполнение ограничений на входные и выходные данные и соотношений между ними, указанные в спецификации модуля. В случае отрицательного результата проверки возбуждается соответствующая исключительная ситуация. Для обработки таких ситуаций в конец этого модуля включаются фрагменты второго рода – обработчики соответствующих исключительных ситуаций. Эти обработчики помимо выдачи необходимой диагностической информации, могут принять меры либо по исключению ошибки в данных (например, потребовать их повторного ввода), либо по ослаблению влияния ошибки (например, во избежание поломки устройств, управляемых с помощью данного ПС, при аварийном прекращении выполнения программы осуществляют мягкую их остановку).

Применение защитного программирования модулей приводит к снижению эффективности ПС как по времени, так и по памяти. Поэтому необходимо разумно регулировать степень применения защитного программирования в зависимости от требований к надежности и эффективности ПС, сформулированных в спецификации качества разрабатываемого ПС. Входные данные разрабатываемого модуля могут поступать как непосредственно от пользователя, так и от других модулей. Наиболее употребительным случаем применения защитного программирования является применение его для первой группы данных, что и означает реализацию устойчивости ПС. Это нужно делать всегда, когда в спецификации качества ПС имеется требование об обеспечении устойчивости ПС. Применение защитного программирования для второй группы входных данных означает попытку обнаружить ошибку в других модулях во время выполнения разрабатываемого модуля, а для выходных данных разрабатываемого модуля - попытку обнаружить ошибку в самом этом модуле во время его выполнения. По существу, это означает частичное воплощение подхода самообнаружения ошибок для обеспечения надежности ПС, о чем шла речь в лекции 3. Этот случай защитного программирования применяется крайне редко - только в том случае, когда требования к надежности ПС чрезвычайно высоки.

11.6. Обеспечение защищенности программных средств

Различают следующие виды защиты ПС от искажения информации:

- защита от сбоев аппаратуры;

- защита от влияния “чужой” программы;
- защита от отказов “своей” программы;
- защита от ошибок оператора (пользователя);
- защита от несанкционированного доступа;
- защита от защиты.

11.6.1 Защита от сбоев аппаратуры. В настоящее время этот вид защиты является не очень злободневной задачей (с учетом уровня достигнутой надежности компьютеров). Но все же полезно знать ее решение. Это обеспечивается организацией т.н. “двойных или тройных просчетов”. Для этого весь процесс обработки данных, определяемый ПС, разбивается по времени на интервалы так называемыми “опорными точками”. Длина этого интервала не должна превосходить половины среднего времени безотказной работы компьютера. В начале каждого такого интервала во вторичную память записывается с некоторой контрольной суммой копия состояния изменяемой в этом процессе памяти (“опорная точка”). Для того, чтобы убедиться, что обработка данных от одной опорной точки до следующей (т.е. один “просчет”) произведена правильно (без сбоев компьютера), производится два таких “просчета”. После первого “просчета” вычисляется и запоминается указанная контрольная сумма, а затем восстанавливается состояние памяти по опорной точке и делается второй “просчет”. После второго “просчета” вычисляется снова указанная контрольная сумма, которая затем сравнивается с контрольной суммой первого “просчета”. Если эти две контрольные суммы совпадают, второй просчет считается правильным, в противном случае контрольная сумма второго “просчета” также запоминается и производится третий “просчет” (с предварительным восстановлением состояния памяти по опорной точке). Если контрольная сумма третьего “просчета” совпадет с контрольной суммой одного из первых двух “просчетов”, то третий просчет считается правильным, в противном случае требуется инженерная проверка компьютера.

11.6.2. Защита от влияния “чужой” программы. При появлении мультипрограммного режима работы компьютера в его памяти может одновременно находиться в стадии выполнения несколько программ, попеременно получающих управление в результате возникающих прерываний (т.н. квазипараллельное выполнение программ). Одна из таких программ (обычно: операционная система) занимается обработкой прерываний и управлением мультипрограммным режимом. Здесь под “чужой” программой понимается программа (или какой-либо программный фрагмент), выполняемая параллельно (или квазипараллельно) по отношению к защищаемой программе (или ее фрагменту). Этот вид защиты должна обеспечить, чтобы эффект выполнения защищаемой программы не зависел от того, какие программы выполняются параллельно с ней, и относится, прежде всего, к функциям операционных систем.

Различают две разновидности этой защиты:

- защита от отказов “чужой” программы,
- защита от злонамеренного влияния “чужой” программы.

Защита от отказов “чужой” программы означает, что на выполнение функций защищаемой программой не будут влиять отказы (проявления ошибок), возникающие в параллельно выполняемых программах. Для того чтобы управляющая программа (операционная система) могла обеспечить защиту себя и других программ от такого влияния, аппаратура компьютера должна реализовывать следующие возможности:

- защиту памяти,
- два режима функционирования компьютера: привилегированный и рабочий (пользовательский),
- два вида операций: привилегированные и ординарные,
- корректную реализацию прерываний и начального включения компьютера,
- временнуе прерывание.

Защита памяти означает возможность программным путем задавать для каждой программы недоступные для нее участки памяти. В привилегированном режиме могут выполняться любые операции (как ординарные, так и привилегированные), а в рабочем режиме - только ординарные. Попытка выполнить привилегированную операцию, а также обратиться к защищенной памяти в рабочем режиме вызывает соответствующее прерывание. К привилегированным операциям относятся операции изменения защиты памяти и режима функционирования, а также доступа к внешней информационной среде. Корректная реализация прерываний и начального включения компьютера означает обязательную установку привилегированного режима и отмену защиты памяти. В этих условиях управляющая программа (операционная система) может полностью защитить себя от влияния отказов других программ. Для этого достаточно, чтобы

- все точки передачи управления при начальном включении компьютера и при прерываниях принадлежали этой программе,
- она не позволяла никакой другой программе работать в привилегированном режиме (при передаче управления любой другой программе должен включаться только рабочий режим),
- она полностью защищала свою память (содержащую, в частности, всю ее управляющую информацию, включая так называемые вектора прерываний) от других программ.

Тогда никто не помешает ей выполнять любые реализованные в ней функции защиты других программ (в том числе и доступа к внешней информационной среде). Для облегчения решения этой задачи часть такой программы помещается в постоянную память. Наличие временнуго прерывания позволяет управляющей программе защититься от зацикливания в других программах (без такого прерывания она могла бы просто лишиться возможности управлять).

Защита от злонамеренного влияния "чужих" программ означает, что изменение внешней информационной среды, предоставленной защищаемой программе, со стороны другой, параллельно выполняемой программы будет невозможно или сильно затруднено без ведома защищаемой программы. Для этого операционная система должна обеспечить подходящий контроль доступа к внешней информационной среде. Необходимым условием обеспечение такого контроля является обеспечения защиты от злонамеренного влияния "чужих" программ хотя бы самой операционной системы. В противном случае такой контроль можно было бы обойти путем изменения операционной системы со стороны "злонамеренной" программы.

Этот вид защиты включает, в частности, и защиту от т.н. "компьютерных вирусов", под которыми понимают фрагменты программ, способные в процессе своего выполнения внедряться (копироваться) в другие программы (или в отдельные программные фрагменты). "Компьютерные вирусы", обладая способностью к размножению (к внедрению в другие программы), при определенных условиях вызывают изменение эффекта выполнения "зараженной" программы, что может привести к серьезным деструктивным изменениям ее внешней информационной среды. Операционная система, будучи защищенной от влияния "чужих" программ, может ограничить доступ к программным фрагментам, хранящимся во внешней информационной среде. Так, например, может быть запрещено изменение таких фрагментов любыми программами, кроме некоторых, которые знает операционная система, или, другой вариант, может быть разрешено только после специальных подтверждений программы (или пользователя).

11.6.3. Защита от отказов "своей" программы. Обеспечивается надежностью ПС, на что ориентирована вся технология программирования, обсуждаемая в настоящем курсе лекций.

11.6.4. Защита от ошибок пользователя. Здесь идет речь не об ошибочных данных, поступающих от пользователя ПС, -защита от них связана с обеспечением устойчивости ПС, а о действиях пользователя, приводящих к деструктивному изменению

состояния внешней информационной среды ПС, несмотря на корректность используемых при этом данных. Защита от таких действий, частично, обеспечивается выдачей предупредительных сообщений о попытках изменить состояние внешней информационной среды ПС с требованием подтверждения этих действий. Для случаев же, когда такие ошибки совершаются, может быть предусмотрена возможность восстановления состояния отдельных компонент внешней информационной среды ПС на определенные моменты времени. Такая возможность базируется на ведении (формировании) архива состояний (или изменений состояния) внешней информационной среды.

11.6.5. Защита от несанкционированного доступа. Каждому пользователю ПС предоставляет определенные информационные и процедурные ресурсы (услуги), причем у разных пользователей ПС предоставленные им ресурсы могут отличаться, иногда очень существенно. Этот вид защиты должен обеспечить, чтобы каждый пользователь ПС мог использовать только то, что ему предоставлено (санкционировано). Для этого ПС в своей внешней информационной среде может хранить информацию о своих пользователях и предоставленным им правах использования ресурсов, а также предоставлять пользователям определенные возможности формирования этой информации. Защита от несанкционированного доступа к ресурсам ПС осуществляется с помощью т.н. *паролей* (секретных слов). При этом предполагается, что каждый пользователь знает только свой пароль, зарегистрированный в ПС этим пользователем. Для доступа к выделенным ему ресурсам он должен предъявить ПС свой пароль. Другими словами, пользователь как бы "вешает замок" на предоставленные ему права доступа к ресурсам, "ключ" от которого имеется только у этого пользователя.

Различают две разновидности такой защиты:

- простая защита от несанкционированного доступа,
- защита от взлома защиты.

Простая защита от несанкционированного доступа обеспечивает защиту от использования ресурсов ПС пользователем, которому не предоставлены соответствующие права доступа или который указал неправильный пароль. При этом предполагается, что пользователь, получив отказ в доступе к интересующим ему ресурсам, не будет предпринимать попыток каким-либо несанкционированным образом обойти или преодолеть эту защиту. Поэтому этот вид защиты может применяться и в ПС, которая базируется на операционной системе, не обеспечивающей полную защиту от влияния “чужих” программ.

Защита от взлома защиты - это такая разновидность защиты от несанкционированного доступа, которая существенно затрудняет преодоление этой защиты. Это связано с тем, что в отдельных случаях могут быть предприняты настойчивые попытки взломать защиту от несанкционированного доступа, если защищаемые ресурсы представляют для кого-то чрезвычайную ценность. Для такого случая приходится предпринимать дополнительные меры защиты. Во-первых, необходимо обеспечить, чтобы такую защиту нельзя было обойти, т. е. должна действовать защита от влияния “чужих” программ. Во-вторых, необходимо усилить простую защиту от несанкционированного доступа использованием в ПС специальных программистских приемов, в достаточной степени затрудняющих подбор подходящего пароля или его вычисление по информации, хранящейся во внешней информационной среде ПС. Использование обычных паролей оказывается недостаточной, когда речь идет о чрезвычайно настойчивом стремлении добиться доступа к ценной информации. Если требуемый пароль (“замок”) в явном виде хранится во внешней информационной среде ПС, то "взломщик" этой защиты относительно легко может его достать, имея доступ к этому ПС. Кроме того, следует иметь в виду, что с помощью современных компьютеров можно осуществлять достаточно большой перебор возможных паролей с целью найти подходящий.

Защититься от этого можно следующим образом. Пароль (секретное слово или просто секретное целое число) X должен быть известен только владельцу защищаемых прав доступа и он не должен храниться во внешней информационной среде ПС. Для проверки прав доступа во внешней информационной среде ПС хранится другое число Y=F(X), однозначно вычисляемое ПС по предъявленному паролю X. При этом функция F может быть хорошо известной всем пользователям ПС, однако она должна обладать таким свойством, что восстановление слова X по Y практически невозможно: при достаточно большой длине слова X (например, в несколько сотен знаков) для этого может потребоваться астрономическое время. Такое число Y будем называть*электронной* (*компьютерной*) *подписью* владельца пароля X (а значит, и защищаемых прав доступа).

Другой способ защиты от взлома защиты связан с защитой сообщений, пересылаемых по компьютерным сетям. Такое сообщение может представлять команду на дистанционный доступ к ценной информации, и этот доступ отправитель сообщения хочет защитить от возможных искажений. Например, при осуществлении банковских операций с использованием компьютерной сети.Использование компьютерной подписи в такой ситуации недостаточно, так как защищаемое сообщение может быть перехвачено “взломщиком” (например, на “перевалочных” пунктах компьютерной сети) и подменено другим сообщением с сохранением компьютерной подписи (или пароля).

Защиту от такого взлома защиты можно осуществить следующим образом [11.4]. Наряду с функцией F, определяющей компьютерную подпись владельца пароля X, в ПС определены еще две функции: Stamp и Notary. При передаче сообщения отправитель, помимо компьютерной подписи Y=F(X), должен вычислить еще другое число S=Stamp(X,R), где X - пароль, а R - текст передаваемого сообщения. Здесь также предполагается, что функция Stamp хорошо известна всем пользователям ПС и обладает таким свойством, что по S практически невозможно ни восстановить число X, ни подобрать другой текст сообщения R с заданной компьютерной подписью Y. При этом передаваемое сообщение (вместе со своей защитой) должно иметь вид:

(R Y S),

причем Y (компьютерная подпись) позволяет получателю сообщения установить истинность клиента, а S как бы скрепляет защищаемый текст сообщения R с компьютерной подписью Y. В связи с этим будем называть число S *электронной*(*компьютерной*) *печатью*. Функция Notary(R,Y,S).проверяет истинность защищаемого сообщения:

(R,Y,S).

Эта позволяет получателю сообщения однозначно установить, что текст сообщения R принадлежит владельцу пароля X.

11.6.6. Защита от защиты. Защита от несанкционированного доступа может создать нежелательную ситуацию для самого владельца прав доступа к ресурсам ПС - он не сможет воспользоваться этими правами, если забудет (или потеряет) свой пароль (“ключ”). Для защиты интересов пользователя в таких ситуациях и предназначена защита от защиты. Для обеспечения такой защиты ПС должно иметь привилегированного пользователя, называемого *администратором ПС.* Администратор ПС должен, в частности, отвечать за функционирование защиты ПС: именно он должен формировать контингент пользователей данного экземпляра ПС, предоставляя каждому из этих пользователей определенные права доступа к ресурсам ПС. В ПС должна быть привилегированная операция (для администратора), позволяющая временно снимать защиту от несанкционированного доступа для пользователя с целью фиксации требуемого пароля (“замка”).