УВАЖАЕМЫЕ СТУДЕНТЫ! Изучите приведенную лекцию, законспектируйте основные тезисы и письменно дайте ответы на контрольные вопросы.

Ответы на вопросы, фотоотчет, предоставить преподавателю на e-mail: r.bigangel@gmail.com до 13.02.2023.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)111-37-59, (Viber, WhatsApp), vk.com: https://vk.com/daykini

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция 29

Тема: **Обеспечение качества программного средства**

Рассматриваемые вопросы:

- Общий обзор.
- Реализация пользовательского интерфейса и обеспечение легкости применения программного средства.
- Обеспечение эффективности программного средства.
- Обеспечение сопровождаемости и управление конфигурацией программного средства.
- Аппаратно-операционные платформы и обеспечение мобильности программного средства.

1. Общая характеристика процесса обеспечения качества программного средства

Как уже отмечалось в лекции 4, спецификация качества определяет основные ориентиры (цели), которые на всех этапах разработки ПС так или иначе влияют при принятии различных решений на выбор подходящего варианта. Однако каждый примитив качества имеет свои особенности такого влияния, тем самым, обеспечение его наличия в ПС может потребовать своих подходов и методов разработки ПС или отдельных его частей. Кроме того, уже отмечалась ранее противоречивость критериев качества ПС, а также и выражающих их примитивов качества: хорошее обеспечение одного какого-либо примитива качества ПС может существенно затруднить или сделать невозможным

обеспечение некоторых других из этих примитивов. Поэтому существенная часть процесса обеспечения качества ПС состоит из поиска приемлемых компромиссов. Эти компромиссы частично должны быть определены уже в спецификации качества ПС: модель качества ПС должна конкретизировать требуемую степень присутствия в ПС каждого его примитива качества и определять приоритеты достижения этих степеней.

Обеспечение качества осуществляется в каждом технологическом процессе: принятые в нем решения в той или иной степени оказывают влияние на качество ПС в целом. В частности и потому, что значительная часть примитивов качества связана не столько со свойствами программ, входящих в ПС, сколько со свойствами документации. В силу отмеченной противоречивости примитивов качества весьма важно придерживаться выбранных приоритетов в их обеспечении. При этом следует придерживаться двух общих принципов:

- сначала необходимо обеспечить требуемую функциональность и надежность ПС, а затем уже доводить остальные критерии качества до приемлемого уровня их присутствия в ПС;
- нет никакой необходимости и, может быть, даже вредно добиваться более высокого уровня присутствия в ПС какого-либо примитива качества, чем тот, который определен в спецификации качества ПС.

Обеспечение функциональности и надежности ПС было рассмотрено в предыдущей лекции. Ниже обсуждается обеспечение других критериев качества ПС.

2. Обеспечение легкости применения программного средства

Легкость применения, в значительной степени, определяется составом и качеством пользовательской документации, а также некоторыми свойствами, реализуемыми программным путем.

С пользовательской документацией связаны такие примитивы качества ПС, как *П-документированность* и *информативность.* Обеспечением ее качества занимаются обычно технические писатели. Этот вопрос будет обсуждаться в следующей лекции. Здесь лишь следует заметить, что там речь будет идти об автономной по отношению к программам документации. В связи с этим следует

обратить внимание на широко используемый в настоящее время подход информирования пользователя в интерактивном режиме (в процессе применения программ ПС). Такое информирование во многих случаях оказывается более удобным для пользователя, чем с помощью автономной документации, так как позволяет пользователю без какого-либо поиска вызывать необходимую информацию за счет использования контекста ее вызова. Такой подход к информированию пользователя является весьма перспективным.

Программным путем реализуются такие примитивы качества ПС как *коммуникабельность*, *устойчивость* и *защищенность*. Обеспечение устойчивости и защищенности уже было рассмотрено в предыдущей лекции. Коммуникабельность обеспечивается соответствующей реализацией обработки исключительных ситуаций и созданием подходящего пользовательского интерфейса.

Возбуждение исключительной ситуации во многих случаях означает, что возникла необходимость информировать пользователя о ходе выполнения программы. При этом выдаваемая пользователю информация должна быть простой для понимания (см. лекцию 4). Однако исключительные ситуации возникают обычно на достаточно низком уровне модульной структуры программы, а создать понятное для пользователя сообщение можно, как правило, на более высоких уровнях этой структуры, где известен контекст, в котором были активизированы действия, приведшие к возникновению исключительной ситуации. Обработку исключительных ситуаций внутри модуля мы уже обсуждали в лекции 8. Для обработки возникшей исключительной ситуации в другом модуле приходится принимать не простые решения. Применяемый часто способ передачи информации о возникшей исключительной ситуации по цепочке обращений к программным модулям (в обратном направлении) является тяжеловесным: он требует дополнительных проверок после возврата из модуля и часто усложняет само обращение к этим модулям за счет задания дополнительных параметров. Приемлемым решением является включение в операционную среду выполнения программ (в *исполнительную поддержку*)

возможностей прямой передачи этой информации обработчикам исключительных ситуаций по динамически формируемой очереди таких обработчиков.

Пользовательский интерфейс представляет средство взаимодействия пользователя с ПС. При разработке пользовательского интерфейса следует учитывать потребности, опыт и способности пользователя [12.1]. Поэтому потенциальные пользователи должны быть вовлечены в процесс разработки такого интерфейса. Большой эффект здесь дает его прототипирование. При этом пользователи должны получить доступ к прототипам пользовательского интерфейса, а их оценка различных возможностей используемого прототипа должна существенно учитываться при создании окончательного варианта пользовательского интерфейса.

В силу большого разнообразия пользователей и видов ПС существует множество различных стилей пользовательских интерфейсов, при разработке которых могут использоваться разные принципы и подходы. Однако следующие важнейшие принципы следует соблюдать всегда [12.1]:

- пользовательский интерфейс должен базироваться на терминах и понятиях, знакомых пользователю;
- пользовательский интерфейс должен быть единообразным;
- пользовательский интерфейс должен позволять пользователю исправлять собственные ошибки;
- пользовательский интерфейс должен позволять получение пользователем справочной информации: как по его запросу, так и генерируемой ПС.

В настоящее время широко распространены командные и графические пользовательские интерфейсы.

Командный пользовательский интерфейс предоставляет пользователю возможность обращаться к ПС с некоторым заданием (запросом), представляемым некоторым текстом (командой) на специальном командном языке (языке заданий). Достоинствами такого интерфейса является возможность его реализации на дешевых алфавитно-цифровых терминалах и возможность минимизации требуемого от пользователя ввода с клавиатуры. Недостатками такого интерфейса являются необходимость изучения командного языка и

достаточно большая вероятность ошибки пользователя при задании команды. В связи с этим командный пользовательский интерфейс обычно выбирают только опытные пользователи. Такой интерфейс позволяет им осуществлять быстрое взаимодействие с компьютером и предоставляет возможность объединять команды в процедуры и программы (см. например, язык Shell операционной системы Unix [12.2]).

Графический пользовательский интерфейс предоставляет пользователю возможности:

- обращаться к ПС путем выбора на экране подходящего графического или текстового объекта,
- получать от ПС информацию на экране в виде графических и текстовых объектов,
- осуществлять прямые манипуляции с графическими и текстовыми объектами, представленными на экране.

Графический пользовательский интерфейс позволяет

- размещать на экране множество различных окон, в которые можно выводить информацию независимо;
- использовать графические объекты, называемые *пиктограммами* (или *иконами*), для обозначения различных информационных объектов или процессов;
- использовать *экранный указатель* для выбора объектов (или их элементов), размещенных на экране; экранный указатель управляется (перемещается) с помощью клавиатуры или мыши.

Достоинством графического пользовательского интерфейса является возможность создания удобной и понятной пользователю модели взаимодействия с ПС (панель управления, рабочий стол и т.п.) без необходимости изучения какого-либо специального языка. Однако его разработка требует больших трудозатрат, сравнимых с трудозатратами по создания самого ПС. Кроме того, возникает серьезная проблема по переносимости ПС на другие операционные системы, так как графический интерфейс существенно зависит от возможностей (*графической*

пользовательской платформы), предоставляемых операционной системой для его создания.

Графический пользовательский интерфейс обобщает такие виды пользовательского интерфейса, как интерфейс типа меню и интерфейс прямого манипулирования.

3. Обеспечение эффективности программного средства

Эффективность ПС обеспечивается принятием подходящих решений на разных этапах его разработки, начиная с разработки его архитектуры. Особенно сильно на эффективность ПС (особенно по памяти) влияет выбор структуры и представления данных. Но и выбор алгоритмов, используемых в тех или иных программных модулях, а также особенности их реализации (включая выбор языка программирования) может существенно повлиять на эффективность ПС. При этом постоянно приходится разрешать противоречие между *временно́й эффективностью* и *эффективностью по памяти* (*ресурсам*). Поэтому весьма важно, чтобы в спецификации качества были явно указаны приоритеты или количественное соотношение между показателями этих примитивов качества. Следует также иметь в виду, что разные программные модули по-разному влияют на эффективность ПС в целом: одни модули могут сильно влиять на временнэю эффективность и практически не влиять на эффективность по памяти, а другие могут существенно влиять на общий расход памяти, не оказывая заметного влияния на время работы ПС. Более того, это влияние (прежде всего, в отношении временнуй эффективности) заранее (до окончания реализации ПС) далеко не всегда можно правильно оценить.

С учетом сказанного, рекомендуется придерживаться следующих принципов для обеспечения эффективности ПС.

- сначала нужно разработать надежное ПС, а потом уж заниматься доведением его эффективности до требуемого уровня в соответствии с его спецификацией качества;
- для повышения эффективности ПС, прежде всего, нужно использовать оптимизирующий компилятор - это может обеспечить требуемую эффективность;

- если эффективность ПС не удовлетворяет спецификации его качества, то найдите самые критические модули с точки зрения требуемой эффективности ПС; эти модули и попытайтесь оптимизировать в первую очередь путем их ручной переделки;
- не следует заниматься оптимизацией модуля, если этого не требуется для достижения требуемой эффективности ПС.

Для отыскания критических модулей с точки зрения временно й эффективности ПС потребуется получить распределение по модулям времени работы ПС путем соответствующих измерений во время выполнения ПС. Это может быть сделано с помощью динамического анализатора (специального программного инструмента), который может определить частоту обращения к каждому модулю в процессе применения ПС.

4. Обеспечение сопровождаемости программного средства

Обеспечение сопровождаемости ПС сводится к обеспечению изучаемости ПС и к обеспечению его модифицируемости.

Изучаемость (подкритерий качества) ПС определяется составом и качеством документации по сопровождению ПС и выражается через такие примитивы качества ПС как *С-документированность*, *информативность*, *понятность*, *структурированность* и *удобочитаемость*. Последние два примитива качества и, в значительной степени, понятность связаны с текстами программных модулей. Вопрос о документации по сопровождению будет обсуждаться в следующей лекции. Здесь мы лишь сделаем некоторые общие рекомендации относительно текстов программ (модулей).

При окончательном оформлении текста программного модуля целесообразно придерживаться следующих рекомендаций, определяющих практически оправданный стиль программирования [12.3, 12.4]:

- используйте в тексте модуля комментарии, проясняющие и объясняющие особенности принимаемых решений; по-возможности, включайте комментарии (хотя бы в краткой форме) на самой ранней стадии разработки текста модуля;

- используйте осмысленные (мнемонические) и устойчиво различимые имена (оптимальная длина имени - 4-12 литер, цифры - в конце), не используйте сходные имена и ключевые слова;
- соблюдайте осторожность в использовании констант (уникальная константа должна иметь единственное вхождение в текст модуля: при ее объявлении или, в крайнем случае, при инициализации переменной в качестве константы);
- не бойтесь использовать необязательные скобки - они обходятся дешевле, чем ошибки;
- размещайте не больше одного оператора в строке; для прояснения структуры модуля используйте дополнительные пробелы (отступы) в начале каждой строки; этим обеспечивается *удобочитаемость* текста модуля;
- избегайте *трюков*, т.е. таких приемов программирования, когда создаются фрагменты модуля, основной эффект которых не очевиден или скрыт (завуалирован), например, побочные эффекты функций.

Структурированность текста модуля существенно упрощает его понимание. *Удобочитаемость* текста модуля может быть обеспечена автоматически путем применения специального программного инструмента - *форматера*.

Модифицируемость (подкритерий качества) ПС определяется, частично, некоторыми свойствами документации, и свойствами, реализуемые программным путем, и выражается через такие примитивы качества ПС как *расширяемость*, *модифицируемость*, *структурированность* и *модульность*.

Расширяемость обеспечивается возможностями автоматически настраиваться на условия применения ПС по информации, задаваемой пользователем. К таким условиям относятся, прежде всего, конфигурация компьютера, на котором будет применяться ПС (в частности, объем и структура его памяти), а также требования конкретного пользователя к функциональным возможностям ПС (например, требования, которые определяют режим применения ПС или конкретизируют структуру информационной среды). К этим возможностям можно отнести и возможность добавления к ПС определенных

компонент. Для реализации таких возможностей в ПС часто включается дополнительная компонента (подсистема), называемая *инсталятором.* Инсталятор осуществляет прием от пользователя необходимой информации и настройку ПС по этой информации. Обычно решение о включении в ПС такой компоненты принимается в процессе разработки архитектуры ПС.

Модифицируемость (примитив качества) обеспечивается такими свойствами документации и свойствами, реализуемые программным путем, которые облегчают внесение изменений и доработок в документацию и программы ПС ручным путем (возможно, с определенной компьютерной поддержкой). В спецификации качества могут быть указаны некоторые приоритетные направления и особенности развития ПС. Эти указания должны быть учтены при разработке архитектуры ПС и модульной структуры его программ. Общая проблема сопровождения ПС - обеспечить, чтобы все его компоненты (на всех уровнях представления) оставались согласованными в каждой новой версии ПС. Этот процесс обычно называют *управлением конфигурацией* (*configuration management*).Чтобы помочь управлению конфигурацией, необходимо, чтобы связи и зависимости между документами и их частями фиксировать в специальной документации по сопровождению [12.5]. Эта проблема усложняется, если в процессе доработки может находиться сразу несколько версий ПС (в разной степени завершенности). Тогда без компьютерной поддержки довольно трудно обеспечить согласованность документов в разных конфигурациях. Поэтому в таких случаях в ПС включается дополнительная компонента (подсистема), называемая *конфигуратором*. С такой компонентой связывают специальную базу данных (или специальный раздел в базе данных), в которой фиксируются связи и зависимости между документами и их частями для всех версий ПС. Обычно решение о включении в ПС такой компоненты принимается в процессе разработки архитектуры ПС. Для обеспечения этого примитива качества в документацию по сопровождению включают специальное руководство, которое описывает, какие части ПС являются аппаратно- и программно-зависимыми, и как возможное развитие ПС учтено в его строении (конструкции).

Структурированность и *модульность* упрощают ручную модификацию программ ПС.

5. Обеспечение мобильности

Проблема мобильности возникает из-за того, что быстрое развитие компьютерной техники и аппаратных средств делает жизненный цикл многих больших программных средств (программных систем) намного продолжительнее периода “морально” оправданного существования компьютеров и аппаратуры, для которых первоначально создавались эти программные средства. Поэтому обеспечение критерия мобильности для таких ПС является весьма важной задачей.

Мобильность ПС определяется такими примитивами качества ПС как *независимость от устройств*, *автономность*, *структурированность* и *модульность*.

Если бы ПС обладало таками примитивами качества, как *независимость от устройств* и *автономность,* и его программы были бы представлены на машинно-независимом языке программирования, то перенос ПС в другую среду обеспечивался бы перетрансляцией (перекомпиляцией) его программ в этой среде. Однако трудно представить реальное ПС, обладающее таким качеством. Тем не менее, таким качеством могут обладать отдельные части программ ПС и даже весьма значительные. А это уже явный намек на то, каким путем следует добиваться мобильности ПС.

Если ПС зависит от устройств (аппаратуры), то в спецификации качества должна быть описана эта компьютерно-аппаратная среда (будем ее называть *аппаратной платформой* [12.6]). Избавится от этой зависимости можно за счет такого примитива качества ПС как автономность. Как правило, ПС строится в рамках некоторой операционной системы (ОС), которая может спрятать специфику аппаратной платформы и, тем самым, сделать ПС независимым от устройств. Но тогда ПС не будет обладать свойством автономности. В этом случае в спецификации качества должна быть описана эта программная среда, над которой строится ПС (будем эту среду называть *операционной платформой* [12.6]). Таким образом, мобильность ПС будет непосредственно связано с мобильностью

используемой ОС: перенос ПС на другую аппаратную платформу осуществляется автоматически, если будет осуществлен перенос на эту платформу используемой ОС. Но обеспечение мобильности ОС является самостоятельной и довольно трудной задачей.

Таким образом, для обеспечения мобильности ПС нужно решить две задачи:

- выделение по возможности наибольшей части программ ПС, обладающей свойствами независимости от устройств и автономности (другими словами, независимой от *аппаратно-операционной платформы*);
- обеспечение сопровождаемости для остальных частей программ ПС.

Для решения этих задач целесообразно выбрать в качестве архитектуры ПС слоистую систему (см. рис. 1). *Основной слой*, реализующий основные функции ПС, должен быть независимым от аппаратно-операционной платформы. Выделяется также слой (часто называемый *ядром* ПС), который включает программные модули, зависящие от аппаратно-операционной платформы. Этот слой должен обеспечивать, в частности, доступ к внешней информационной среде ПС. Между этими слоями должен быть определен интерфейс, независимый от аппаратно-операционной платформы и обеспечивающий правила обращения из основного слоя к модулям ядра. Будем называть этот интерфейс *системным*. Использование графических пользовательских интерфейсов требует выделение еще одного программного слоя, зависящего от той части аппаратно-операционной платформы (*графической пользовательской платформы*), на которой строятся пользовательские интерфейсы. Будем называть этот слой *оболочкой* ПС. Между оболочкой и основным слоем также должен быть определен интерфейс, независимый от графической пользовательской платформы и обеспечивающий правила обращения из оболочки к модулям основного слоя.

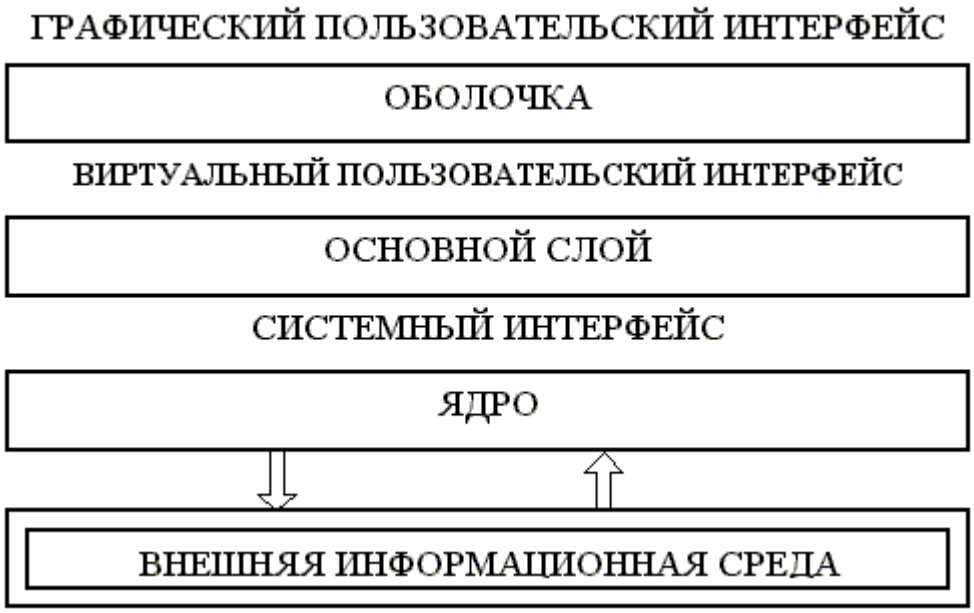

Рис. 1. Рекомендуемая архитектура мобильного ПС

Модульность ПС позволяет сформировать указанные слои, выделяя программные модули с требуемыми свойствами и распределяя их между указанными слоями. *Модульность* и *структурированность* оболочки и ядра позволяют обеспечить эти слои свойством модифицируемости. При этом желательно, чтобы каждый модуль этих слоев был ориентирован на реализацию каких-либо функций управления четко выделенной компоненты аппаратно-операционной среды. Для этого используются такие методы как унификация интерфейсов, стандартизация протоколов и т.п. [12.6].

Контрольные вопросы

1. Какие задачи приходиться решать при обеспечении коммуникабельности ПС?

2. Какие возможности предоставляет пользователю графический пользовательский интерфейс?

3. Как нужно действовать для обеспечения эффективности ПС?

4. Что такое инсталятор программного средства (ПС)?

5. Что такое управление конфигурацией ПС?

6. Что такое ядро ПС?

7. Что такое оболочка ПС?