Информация для размещения на официальном сайте ГБПОУ «Светлоградский региональный сельскохозяйственный колледж»

Для электронного обучения

Группа	324
Дата	26.02.2025
Время	8.10 – 9.30
Наименование УД/МДК/УП/ПП	МДК 01.02 Поддержка и тестирование программных модулей
Ф.И.О. преподавателя	Коваленко Аркадий Владимирович
Электронная почта	Aricus2007@inbox.ru
Основная литература	**Основные источники:** Поддержка и тестирование программных модулей. Емелина Е.И. 2024 год учебник, КноРус, https://book.ru/books/954267
Тема	Спецификация программного модуля
Задание(лекция)	**Спецификация программного модуля** содержит, во-первых, синтаксическую спецификацию его входов, позволяющую построить на используемом языке программирования правильное обращение к нему (к любому его входу), и, во-вторых, функциональную спецификацию модуля (описание семантики функций, выполняемых этим модулем по каждому из его входов). Основные характеристики программного модуля ***Размер модуля*** измеряется числом содержащихся в нем операторов (строк).Модуль не должен быть слишком маленьким или слишком большим. Маленькие модули приводят к громоздкой модульной структуре программы и могут не окупать накладных расходов, связанных с их оформлением. Большие модули неудобны для изучения и изменений, они могут существенно увеличить суммарное время повторных трансляций программы при отладке программы. Обычно рекомендуются программные модули размером от нескольких десятков до нескольких сотен операторов. ***Прочность модуля*** - это мера его внутренних связей. Чем выше прочность модуля, тем больше связей он может спрятать от внешней по отношению к нему части программы и, следовательно, тем больший вклад в упрощение программы он может внести. Самой слабой степенью прочности обладает модуль, *прочный по совпадению*. Это такой модуль, между элементами которого нет осмысленных связей. Такой модуль может быть выделен, например, при обнаружении в разных местах программы повторения одной и той же последовательности операторов, которая и оформляется в отдельный модуль. Необходимость изменения этой последовательности в одном из контекстов может привести к изменению этого модуля, что может сделать его использование в других контекстах ошибочным. Такой класс программных модулей не рекомендуется для использования. ***Функционально прочный модуль*** - это модуль, выполняющий (реализующий) одну какую-либо определенную функцию. При реализации этой функции такой модуль может использовать и другие модули. Такой класс программных модулей рекомендуется для использования. **Информационно прочный модуль** - это модуль, выполняющий (реализующий) несколько операций (функций) над одной и той же структурой данных (информационным объектом), которая считается неизвестной вне этого

	модуля. Для каждой из этих операций в таком модуле имеется свой вход со своей формой обращения к нему. Такой класс следует рассматривать как класс программных модулей с высшей степенью прочности. Информационно-прочный модуль может реализовывать, например, абстрактный тип данных. **Сцепление модуля** - это мера его зависимости по данным от других модулей. Характеризуется способом передачи данных. Чем слабее сцепление модуля с другими модулями, тем сильнее его независимость от других модулей. **Рутинность модуля** - это его независимость от предыстории обращений к нему. Модуль будем называть рутинным, если результат (эффект) обращения к нему зависит только от значений его параметров (и не зависит от предыстории обращений к нему). Модуль будем называть зависящим от предыстории, если результат (эффект) обращения к нему зависит от внутреннего состояния этого модуля, хранящего следы предыдущих обращений к нему. Порядок разработки программного модуля При разработке программного модуля целесообразно придерживаться следующего порядка: 1) изучение и проверка спецификации модуля, выбор языка программирования; 2) выбор алгоритма и структуры данных; 3) программирование модуля; 4) шлифовка текста модуля; 5) проверка модуля; 6) компиляция модуля. ***Первый шаг*** разработки программного модуля в значительной степени представляет собой смежный контроль структуры программы снизу: изучая спецификацию модуля, разработчик должен убедиться, что она ему понятна и достаточна для разработки этого модуля. В завершении этого шага выбирается язык программирования. ***На втором шаге*** разработки программного модуля необходимо выяснить, не известны ли уже какие-либо алгоритмы для решения поставленной и/или близкой к ней задачи. И если найдется подходящий алгоритм, то целесообразно им воспользоваться. Выбор подходящих структур данных, которые будут использоваться при выполнении модулем своих функций, в значительной степени предопределяет логику и качественные показатели разрабатываемого модуля, поэтому его следует рассматривать как весьма ответственное решение. ***На третьем шаге*** осуществляется построение текста модуля на выбранном языке программирования. Обилие всевозможных деталей, которые должны быть учтены при реализации функций, указанных в спецификации модуля, легко могут привести к созданию весьма запутанного текста, содержащего массу ошибок и неточностей. Искать ошибки в таком модуле и вносить в него требуемые изменения может оказаться весьма трудоемкой задачей. ***Следующий шаг*** разработки модуля связан с приведением текста модуля к завершенному виду в соответствии со спецификацией качества ПС. При программировании модуля разработчик основное внимание уделяет правильности реализации функций модуля, оставляя недоработанными комментарии и допуская некоторые нарушения требований к стилю программы. При шлифовке текста модуля он должен отредактировать имеющиеся в тексте комментарии и, возможно, включить в него дополнительные комментарии с целью обеспечить требуемые примитивы качества. С этой же целью производится редактирование текста программы для выполнения стилистических требований.

	Шаг проверки модуля представляет собой ручную проверку внутренней логики модуля до начала его отладки (использующей выполнение его на ЭВМ), реализует общий принцип, сформулированный для технологии программирования. И, наконец, ***последний шаг*** разработки модуля означает переход к процессу отладки модуля. **Контроль структуры программы.** Для контроля структуры программы можно использовать три метода: • статический контроль, • смежный контроль, • сквозной контроль. ***Статический контроль*** состоит в оценке структуры программы сточки зрения хорошо ли программа разбита на модули с учетом значений рассмотренных выше основных характеристик модуля. ***Смежный контроль сверху*** - это контроль со стороны разработчиков архитектуры и внешнего описания ПС. ***Смежный контроль снизу*** - это контроль спецификации модулей со стороны разработчиков этих модулей. ***Сквозной контроль*** - это мысленное прокручивание (проверка) структуры программы при выполнении заранее разработанных тестов. Является видом динамического контроля так же, как и ручная имитация функциональной спецификации или архитектуры ПС.
Закрепление знаний(контроль)	Сделать конспект лекции и предоставить 5.03.2025

Дата 26.02.2025 _____ Коваленко А. В.

Подпись Ф.И.О. преподавателя

Информация для размещения на официальном сайте ГБПОУ «Светлоградский региональный сельскохозяйственный колледж»

Для электронного обучения

Группа	324
Дата	27.02.2025
Время	8.10 – 9.30
Наименование УД/МДК/УП/ПП	МДК 01.02 Поддержка и тестирование программных модулей
Ф.И.О. преподавателя	Коваленко Аркадий Владимирович
Электронная почта	Aricus2007@inbox.ru
Основная литература	**Основные источники:** Поддержка и тестирование программных модулей. Емелина Е.И. 2024 год учебник, КноРус, https://book.ru/books/954267
Тема	Рефакторинг программного кода. Методы организации рефакторинга и оптимизации кода
Задание(лекция)	Рефакторинг — это переработка исходного кода программы, чтобы он стал более простым и понятным. Рефакторинг не меняет поведение программы, не исправляет ошибки и не добавляет новую функциональность. Он делает код более понятным и удобочитаемым. Зачем нужен рефакторинг Стройный, хорошо структурированный код легко читается и быстро дорабатывается. Но редко удаётся сразу сделать его таким. Разработчики спешат, в процессе могут меняться требования к задаче, тестировщики находят баги, которые нужно быстро исправить, или возникают срочные доработки, и их приходится делать второпях. В результате даже изначально хорошо структурированный исходник становится беспорядочным и непонятным. Программисты знают, как легко завязнуть в этом хаосе. Причём неважно, чужой это код или собственный. Чтобы решить все эти проблемы, делается рефакторинг программы. В новом проекте он нужен, чтобы: • сохранить архитектуру проекта, не допустить потери структурированности; • упростить будущую жизнь разработчиков, сделать код понятным и прозрачным для всех членов команды; • ускорить разработку и поиск ошибок. Чем рефакторинг отличается от оптимизации Рефакторинг — не оптимизация, хотя и может быть с нею связан. Часто его проводят одновременно с оптимизацией, поэтому понятия кажутся синонимами. Но у этих процессов разные цели. Цель оптимизации — улучшение производительности программы, а рефакторинга — улучшение понятности кода. После оптимизации исходный код может стать сложнее для понимания. После рефакторинга программа может начать работать быстрее, но главное — её код становится проще и понятнее. Когда нужно срочно улучшать код Признаки, показывающие, что назрела необходимость в рефакторинге:

	• Программа работает, но даже небольшие доработки сильно затягиваются из-за того, что каждый раз приходится долго разбираться в коде. • Разработчик постоянно не может точно сказать, сколько времени ему нужно на выполнение задачи, потому что “там надо вначале разбираться”. • Одинаковые изменения приходится вносить в разные места текста программы. Как делают рефакторинг Рефакторинг — это маленькие последовательные улучшения кода. Чистить можно всё, но в первую очередь найдите эти проблемы: **Мёртвый код.** Переменная, параметр, метод или класс больше не используются: требования к программе изменились, но код не почистили. Мёртвый код может встретиться и в сложной условной конструкции, где какая-то ветка никогда не исполняется из-за ошибки или изменения требований. Такие элементы или участки текста нужно удалить. **Дублирование.** Один и тот же код выполняет одно и то же действие в нескольких местах программы. Вынесите эту часть в отдельную функцию. **Имена переменных, функций или классов не передают их назначение.** Имена должны сообщать, почему элемент кода существует, что он делает и как используется. Если видите, что намерения программиста непонятны без комментария, — рефакторьте. **Слишком длинные функции и методы.** Оптимальный размер этих элементов — 2-3 десятка строк. Если получается больше, разделите функцию на несколько маленьких и добавьте одну общую. Пусть маленькие выполняют по одной операции, а общая функция их вызывает. **Слишком длинные классы.** То же самое. Оптимальная длина класса — 20–30 строк. Разбейте длинный класс на несколько маленьких и включите их объекты в один общий класс. **Слишком длинный список параметров функции или метода.** Они только запутывают, а не помогают. Если все эти параметры действительно нужны, вынесите их в отдельную структуру или класс с понятным именем, а в функцию передайте ссылку на него. **Много комментариев.** Плохой код часто прикрывается обильными комментариями. Если почувствовали желание пояснить какой-то участок кода, попробуйте сначала его переписать, чтобы и так стал понятным. Бесполезные комментарии загромождают программу, а устаревшие и неактуальные вводят в заблуждение. В чём опасности рефакторинга Мы всё-таки меняем рабочий код. Тут можно не только всё упростить, но и сильно напортачить. Небрежный рефакторинг может отбросить выполнение проекта на дни и недели.
Закрепление знаний(контроль)	Сделать конспект лекции и предоставить 5.03.2025

Дата 27.02.2025 _____ Коваленко А. В.

Подпись Ф.И.О. преподавателя