|  |  |  |
| --- | --- | --- |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 16.05.2022,  17.05.2022 | **Практическая работа №17-18**  Тема: Проектирование интерфейса.  Наименование работы: Проектирование интерфейса разрабатываемого программного продукта. Создание макета. Проектирование структуры сайта. Создание макета.  Цель: закрепление умений и навыков по разработке макета пользовательского интерфейса.  **Задание 1**  Изучить и законспектировать материал, ответить на контрольные вопросы.  Золотое сечение — это самая комфортная для глаза пропорция, форма, в основе построения которой лежит сочетание симметрии и золотого сечения, способствует наилучшему зрительному восприятию и появлению ощущения красоты и гармонии.  Применяя принцип кошелька Миллера в дизайне интерфейсов, следует группировать элементы в программе (кнопки на панелях инструментов, пункты меню, закладки, опции на этих закладках и т. п.) с учетом этого правила— т. е. не более семи в группе, в крайнем случае — девяти.  Принцип группировки  Согласно этому правилу, экран программы должен быть разбит на ясно очерченные блоки элементов, может быть, даже с заголовком для каждого блока. При этом группировка, естественно, должна быть осмысленной: как расположение элементов в группах, так и расположение самих групп друг от друга должны быть продуманы.  Философский принцип, носящий название "Бритва Оккама", гласит: "Не множить сущности без надобности". Или, как говорят американцы, KISS ("Keep It Simple, Stupid" — "He усложняй, болван").  Заимствование широко распространенных приемов дизайна интерфейсов и удачных находок авторов конкурирующих программ позволяет резко сократить время обучения и повысить комфорт пользователя. При работе он будет использовать уже приобретенные навыки — этот вопрос затрагивает и принцип равенства между системой и реальным миром.  Контрольные вопросы:   1. Золотое сечение, кошелек Миллера, принцип группировки, бритва Оккама или KISS, видимость отражает полезность, умное заимствование. 2. Золотое сечение — это самая комфортная для глаза пропорция, форма, в основе построения которой лежит сочетание симметрии и золотого сечения, способствует наилучшему зрительному восприятию и появлению ощущения красоты и гармонии. 3. Применяя принцип кошелька Миллера в дизайне интерфейсов, следует группировать элементы в программе (кнопки на панелях инструментов, пункты меню, закладки, опции на этих закладках и т. п.) с учетом этого правила— т. е. не более семи в группе, в крайнем случае — девяти. 4. Согласно этому правилу, экран программы должен быть разбит на ясно очерченные блоки элементов, может быть, даже с заголовком для каждого блока. При этом группировка, естественно, должна быть осмысленной: как расположение элементов в группах, так и расположение самих групп друг от друга должны быть продуманы. 5. Философский принцип, носящий название "Бритва Оккама", гласит: "Не множить сущности без надобности". Или, как говорят американцы, KISS ("Keep It Simple, Stupid" — "He усложняй, болван"). 6. Заимствование широко распространенных приемов дизайна интерфейсов и удачных находок авторов конкурирующих программ позволяет резко сократить время обучения и повысить комфорт пользователя. При работе он будет использовать уже приобретенные навыки — этот вопрос затрагивает и принцип равенства между системой и реальным миром.   **Задание 2**  Разработал макет пользовательского интерфейса.  (Приложение 15.1)  **Задание 3**  Описал и показал на примере разработанного макета, какие принципы проектирования интерфейса были использованы в моем случае.  В моем программном средстве используется принципы такие как: Бритва Оккама, Принцип группировки, Видимость отражает полезность.  **Задание 4**  Ответил на вопросы теста: https://onlinetestpad.com/ 5ckzm72ujalew. |  |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 18.05.2022 | **Практическая работа №19**  Тема: Шаблоны проектирования.  Наименование работы: Использование шаблонов при разработке программного продукта.  Цель: ознакомиться с понятием шаблоны проектирования; изучить их классификации, рассмотреть примеры использования шаблонов проектирования в конкретной ситуации.  **Задание 1**  Изучил теоретический материал, представленный в документе ‘Шаблоны проектирования.pdf’. Законспектировал в дневнике-отчете определение, классификацию паттернов, каждому виду дайте характеристику.  Паттерны проектирования и их представление в нотации UML   1. **Паттерны, их классификация**   При реализации проектов по разработке программных систем и моделированию бизнес-процессов встречаются ситуации, когда решение проблем в различных проектах имеют сходные структурные черты. Попытки выявить похожие схемы или структуры в рамках *объектно-ориентированного анализа и проектирования* привели к появлению понятия паттерна, которое из абстрактной категории превратилось в непременный *атрибут* современных CASE-средств  Паттерны ООАП различаются степенью детализации и уровнем абстракции. Предлагается следующая общая классификация паттернов по категориям их применения:   * *Архитектурные паттерны* * *Паттерны проектирования* * *Паттерны анализа* * *Паттерны тестирования* * *Паттерны реализации*   ***Архитектурные паттерны (Architectural patterns)*** - множество предварительно определенных подсистем со спецификацией их ответственности, правил и базовых принципов установления отношений между ними.  *Архитектурные паттерны* предназначены для спецификации фундаментальных схем структуризации программных систем. Наиболее известными паттернами этой категории являются паттерны GRASP (*General* *Responsibility* *Assignment* *Software* *Pattern*). Эти паттерны относятся к уровню системы и подсистем, но не к уровню классов. Как правило, формулируются в обобщенной форме, используют обычную терминологию и не зависят от области приложения. Паттерны этой категории систематизировал и описал К. Ларман.  ***Паттерны проектирования (Design patterns)*** - специальные схемы для уточнения структуры подсистем или компонентов программной системы и отношений между ними.  *Паттерны проектирования* описывают общую структуру взаимодействия элементов программной системы, которые реализуют исходную проблему проектирования в конкретном контексте. Наиболее известными паттернами этой категории являются паттерны GoF (*Gang* of Four), названные в честь Э. Гаммы, Р. Хелма, Р. Джонсона и Дж. Влиссидеса, которые систематизировали их и представили общее описание. Паттерны GoF включают в себя 23 паттерна. Эти паттерны не зависят от языка реализации, но их реализация зависит от области приложения.  ***Паттерны анализа (Analysis patterns)*** - специальные схемы для представления общей организации процесса моделирования.  *Паттерны анализа* относятся к одной или нескольким предметным областям и описываются в терминах *предметной области*. Наиболее известными паттернами этой группы являются паттерны бизнес-моделирования *ARIS* (*Architecture* of Integrated *Information* Systems), которые характеризуют абстрактный *уровень представления* бизнес-процессов. В дальнейшем *паттерны анализа* конкретизируются в типовых моделях с целью выполнения аналитических оценок или имитационного моделирования бизнес-процессов.  ***Паттерны тестирования (Test patterns)*** - специальные схемы для представления общей организации процесса тестирования программных систем.  К этой категории паттернов относятся такие паттерны, как тестирование черного ящика, белого ящика, отдельных классов, системы. Паттерны этой категории систематизировал и описал М. Гранд. Некоторые из них реализованы в инструментальных средствах, наиболее известными из которых является *IBM* Test Studio. В связи с этим *паттерны тестирования* иногда называют стратегиями или схемами тестирования.  ***Паттерны реализации (Implementation patterns)*** - совокупность компонентов и других элементов реализации, используемых в структуре модели при написании программного кода.  Эта категория паттернов делится на следующие подкатегории: паттерны организации программного кода, паттерны *оптимизации программного кода*, паттерны устойчивости кода, паттерны разработки графического интерфейса пользователя и др. Паттерны этой категории описаны в работах М. Гранда, К. Бека, Дж. Тидвелла и др. Некоторые из них реализованы в популярных интегрированных средах программирования в форме шаблонов создаваемых проектов. В этом случае выбор шаблона программного приложения позволяет получить некоторую заготовку программного кода.  **Задание 2**  Изучил статью «Практика применения паттернов проектирования.pdf».  **Задание 3**  Нашлё информацию в Интернете и законспектировал в дневник-отчет описание 3 шаблонов (паттернов) проектирования.   1. Посетитель   Посетитель — это поведенческий паттерн проектирования, который позволяет добавлять в программу новые операции, не изменяя классы объектов, над которыми эти операции могут выполняться.  Ваша задача — сделать экспорт этого графа в XML. Дело было бы плёвым, если бы вы могли редактировать классы узлов. Достаточно было бы добавить метод экспорта в каждый тип узла, а затем, перебирая узлы графа, вызывать этот метод для каждого узла. Благодаря полиморфизму, решение получилось бы изящным, так как вам не пришлось бы привязываться к конкретным классам узлов.  Но, к сожалению, классы узлов вам изменить не удалось. Системный архитектор сослался на то, что код классов узлов сейчас очень стабилен, и от него многое зависит, поэтому он не хочет рисковать и позволять кому-либо его трогать.  Ваша задача — сделать экспорт этого графа в XML. Дело было бы плёвым, если бы вы могли редактировать классы узлов. Достаточно было бы добавить метод экспорта в каждый тип узла, а затем, перебирая узлы графа, вызывать этот метод для каждого узла. Благодаря полиморфизму, решение получилось бы изящным, так как вам не пришлось бы привязываться к конкретным классам узлов.  Но, к сожалению, классы узлов вам изменить не удалось. Системный архитектор сослался на то, что код классов узлов сейчас очень стабилен, и от него многое зависит, поэтому он не хочет рисковать и позволять кому-либо его трогать.  Применимость   Когда вам нужно выполнить какую-то операцию над всеми элементами сложной структуры объектов, например, деревом.   Посетитель позволяет применять одну и ту же операцию к объектам различных классов.   Когда над объектами сложной структуры объектов надо выполнять некоторые не связанные между собой операции, но вы не хотите «засорять» классы такими операциями.   Посетитель позволяет извлечь родственные операции из классов, составляющих структуру объектов, поместив их в один класс-посетитель. Если структура объектов является общей для нескольких приложений, то паттерн позволит в каждое приложение включить только нужные операции.   Когда новое поведение имеет смысл только для некоторых классов из существующей иерархии.   Посетитель позволяет определить поведение только для этих классов, оставив его пустым для всех остальных.   1. Наблюдатель   В основе этого шаблона проектирования лежит отношение «один ко многим» между многочисленными объектами. Это позволяет вам настроить механизм подписки, который позволяет другим объектам получать оповещения о каждом появлении объекта, на который вы подписаны.  Kafka, RabbitMQ, Amazon SNS и NATS — это некоторые реальные примеры систем публикации/подписки, которые реализуют шаблон издатель/подписчик (вариант Observer).  Ниже приведены некоторые примеры работы этого шаблона: В мире веб-разработки, особенно с React, вы слышали о Redux для управления состоянием вашего приложения. Redux — это реализация шаблона Наблюдатель. Когда вы прикрепляете действие для обновления состояния хранилища, компоненты, прослушивающие изменения, соответствующим образом корректируют свои представления.  Если код отправлен в удаленный репозиторий, среда CI отслеживает его изменения и выполняет сборку.  Процедурное программирование, управляемое событиями, используется для имитации шаблона Наблюдатель. Как и другие шаблоны проектирования, этот шаблон позволяет нам определять слабосвязанные системы. Используя эту технику, мы можем разрабатывать поддерживаемое и модульное программное обеспечение. Мы также можем достичь очевидной сегментации между отдельными участниками в системе, управляемой событиями.   1. Прототип   Прототип — это порождающий паттерн проектирования, который позволяет копировать объекты, не вдаваясь в подробности их реализации.  Паттерн Прототип поручает создание копий самим копируемым объектам. Он вводит общий интерфейс для всех объектов, поддерживающих клонирование. Это позволяет копировать объекты, не привязываясь к их конкретным классам. Обычно такой интерфейс имеет всего один метод clone.  Реализация этого метода в разных классах очень схожа. Метод создаёт новый объект текущего класса и копирует в него значения всех полей собственного объекта. Так получится скопировать даже приватные поля, так как большинство языков программирования разрешает доступ к приватным полям любого объекта текущего класса.  Объект, который копируют, называется прототипом (откуда и название паттерна). Когда объекты программы содержат сотни полей и тысячи возможных конфигураций, прототипы могут служить своеобразной альтернативой созданию подклассов.  Контрольные вопросы:   1. Шаблон проектирования или паттерн в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста. 2. Шаблоны проектирования описывают подходы к решению кон кретных задач. Детали реализации могут существенно меняться в зависимости от более широкого контекста. От этого контекста зависит выбор используемого языка программирования, природа приложения, размер проекта и специфика задачи. 3. Паттерны отличаются по уровню сложности, детализации и охвата проектируемой системы. Проводя аналогию со строительством, вы можете повысить безопасность перекрёстка, поставив светофор, а можете заменить перекрёсток целой автомобильной развязкой с подземными переходами.   Самые низкоуровневые и простые паттерны — идиомы. Они не универсальны, поскольку применимы только в рамках одного языка программирования.  Самые универсальные — архитектурные паттерны, которые можно реализовать практически на любом языке. Они нужны для проектирования всей программы, а не отдельных её элементов.   1. Шаблон фасад (англ. Facade) — структурный шаблон проектирования, позволяющий скрыть сложность системы путём сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы. 2. Паттерн проектирования в контексте языка UML представляет собой параметризованную кооперацию вместе с описанием базовых принципов ее использования. |  |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 19.05.2022 | **Практическая работа №20**  Тема: Обоснование выбора средств реализации проекта. Кодирование модулей разрабатываемого программного обеспечения.  Наименование работы: Обоснование выбора средств реализации проекта.  Цель: изучить современную технологию создания приложения.  **Задание 1**  Изучил теоретический материал по https://technologiarpo.blogspot.com/p/28\_27.html. Проанализировал какую из сред разработки целесообразно использовать для реализации вашего индивидуального задания. Обосновать свой выбор.  Целевой платформой для моего проекта будет любая платформа с поддержкой браузеров, и работой интернета.  Время исполнения проекта соответствует графику практических работ.  Будут использоваться следующие языки программирования: C# так же для создания базы данных будут использованы: phpMyAdmin.  Так же будет проведено тестирование, а именно: Тестирование безопасности, Функциональное тестирование, Тестирование удобства пользования.  **Задание 2**  Проанализировал подключение разработанной вами БД к приложению. В отчет привел список компонентов и их настроек, которые необходимо использовать при подключении к БД, а также при разработке интерфейса согласно примера:   |  |  | | --- | --- | | **Компоненты** | **Описание** | | Label | Используется для размещения на формах и других контейнерах текста, который не изменяется пользователем. Компонент визуальный. | | Button | Используется для создания кнопок, которыми пользователь выбирает команды в приложении. | | TextBox | Используется пользователями для ввода запрашиваемых данных | | SaveFileDialog | Используется системой для выполнения действия сохранения. | | BindingNavigator | Используется для навигации по таблице. | | dataGridView | Используется для отображения таблицы, и работы с ней | | openFileDialog | Используется системой для выполнения действия открытие. | | MenuStrip | Используется для удобного перемещения по приложению. | | TabControl | Помогает пользователю переключаться среди таблиц | | contextMenuStrip | Меню позволяющее пользователю быстро ориентироваться |   **Задание 3**  Разработал интерфейс ПС, разместил все компоненты на формы. В отчет вставил скриншоты форм. (Приложение 16.1) |  |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 20.05.2022,  21.05.2022 | **Практическая работа №21-22**  Тема: Обоснование выбора средств реализации проекта. Кодирование модулей разрабатываемого программного обеспечения.  Наименование работы: Кодирование основных модулей программной системы. Кодирование дополнительных модулей программной системы.  Цель: закрепление умений и навыков по кодированию модулей программного средства, корректировке кода готового ПС.  **Часть 1**  **Задание 1**  Изучил теоретический материал «Методы программирования.pdf». Законспектировал методы и их характеристики в дневнике-отчете (не менее 2 страниц).  Модульное программирование  Приступая к разработке каждой программы, следует иметь в виду, что она, как правило, является большой системой, поэтому надо принять меры для ее упрощения. Для этого такую программу разрабатывают по частям, которые называются программными модулями. А сам такой метод разработки программ называют модульным программирование. Программный модуль – это любой фрагмент описания процесса, оформляемый как самостоятельный программный продукт, пригодный для использования в описаниях процесса. Это означает, что каждый программный модуль программируется, компилируется и отлаживается отдельно от других модулей программы, и тем самым, физически разделен с другими модулями программы. Таким образом, программный модуль может рассматриваться и как средство борьбы со сложностью программ, и как средство борьбы с дублированием в программировании.  Модульное программирование является воплощением в процессе разработки программ обоих общих методов борьбы со сложностью: и обеспечение независимости компонент системы и использование иерархических структур. Для воплощения первого метода формулируются определенные требования, которым должен удовлетворять программный модуль, т.е. выявляются основные характеристики «хорошего» программного модуля. Для воплощения второго метода используют древовидные модульные структуры программ (включая деревья со сросшимися ветвями).  Основные характеристики программного модуля  Не всякий программный модуль способствует упрощению программы. Выделить хороший с этой точки зрения модуль является серьезной творческой задачей.  Для оценки приемлемости выделенного модуля используются некоторые критерии. Так, Хольт предложил следующие два общих таких критерия:  • хороший модуль снаружи проще, чем внутри;  • хороший модуль проще использовать, чем построить. Майерс предлагает для оценки приемлемости программного модуля использовать более конструктивные его характеристики:  • размер модуля;  • прочность модуля;  • сцепление с другими модулями;  • рутинность модуля (независимость от предыстории обращений к нему).  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.  Проектирование модуля  Модульное проектирование относится к процессу расчленения больших проблем на более узкие, более управляемые под проблемы. Первым шагом проектирования является решение, в каком месте должна быть граница между этими под проблемами. Для получения максимальных преимуществ от использования модульного программирования каждая подпроблема или модуль должны иметь один вход и один выход. В этом случае можно легко отслеживать поток управления в программе. В любом месте модуля должна иметься возможность увидеть точку входа в модуль и определить точное значение переменных и регистров в этой точке, а затем проследить функционирование модуля без тревоги об искажении программы. Один вход обеспечивает возврат потока управления в точку вызова. По этой причине, модульные программы почти всегда выполняются как структуры «CALL-RET».  Использование нескольких предложений RET в модуле не должно нарушать правило одного входа, поскольку все инструкции RET возвращают управление в одну и туже точку. Точно также, переход к общему RET в конце модуля, не изменяет его структуру, а добавляет лишь коды и увеличивает его сложность. С другой стороны, вход или выход из модуля не по этому правилу перечеркивает наибольшие преимущества модульного программирования: ясность и удобство сопровождения.  Имеется исключение из правила входа в модуль. Это происходит при использовании таблицы переходов для реализации потока управления внутри программы. Таблица перехода используется путем «проталкивания» адреса возврата в стек, вычисления индекса требуемого адреса перехода в таблице и выполнения перехода в памяти. При практическом выполнении декомпозиции модулей можно самим найти некоторое количество альтернативных решений. Прежде чем осуществить правильный выбор, необходимо знать альтернативы. Цель состоит в выборе таких вариантов, которые создадут наилучшие условия проектирования.  Методы разработки структуры модульной программы  Как уже отмечалось выше, в качестве модульной структуры программы принято использовать древовидную структуру, включая деревья со сросшимися ветвями. В узлах такого дерева размещаются программные модули, а направленные дуги (стрелки) показывают статическую подчиненность модулей, т.е. каждая дуга показывает, что в тексте модуля, из которого она исходит, имеется ссылка на модуль, в который она входит. Другими словами, каждый модуль может обращаться к подчиненным ему модулям, т.е. выражается через эти модули. При этом модульная структура программы, в конечном счете, должна включать и совокупность спецификаций модулей, образующих эту программу. Спецификация программного модуля содержит:  • синтаксическую спецификацию его входов, позволяющую построить на используемом языке программирования синтаксически правильное обращение к нему (к любому его входу);  • функциональную спецификацию модуля (описание семантики функций, выполняемых этим модулем по каждому из его входов).  В процессе разработки программы ее модульная структура может по – разному формироваться и использоваться для определения порядка программирования и отладки модулей, указанных в этой структуре. Поэтому можно говорить о разных методах разработки структуры программы. Обычно в литературе обсуждаются два метода: метод восходящей разработки и метод нисходящей разработки.  **Задание 2**  Разработал кодирование пользовательского меню. Скриншоты меню, а также листинг кода вставил в «Приложение 17.1».  **Задание 3**  Кодирование пользовательского интерфейса. Скриншоты разработанных форм вставил в «Приложение 17.2». Листинг кода (можно скриншот) вставьте в «Приложение 17.3».  **Часть 2**  **Задание 1**  Разработал формы приложения, их кодирование. Скриншоты разработанных форм вставила в «Приложение 17.2». Листинг кода (можно скриншот) вставьте в «Приложение 17.3».  **Задание 2**  Разработка оставшихся форм приложения, их кодирование. Скриншоты разработанных форм вставила в «Приложение 17.2». Листинг кода (можно скриншот) вставила в «Приложение 17.3». |  |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 23.05.2022,  24.05.2022 | **Практическая работа №23-24**  Тема: Обоснование выбора средств реализации проекта. Кодирование модулей разрабатываемого программного.  Наименование работы: Отладка модулей программной системы. Интегрирование программных модулей.  Цель: закрепление умений и навыков по кодированию, отладке и интегрированию модулей программного средства, корректировке кода готового ПС.  **Задание 1**  Изучил теоретический материал. Законспектировал основные определения.  Отладка - это некоторый детективный процесс. Программа, в которую внесены изменения, подозревается в том, что она работает некорректно. Презумпция невиновности здесь не применима.  Локализацией называют процесс определения оператора программы, выполнение которого вызвало нарушение нормального вычислительного процесса. Для исправления ошибки необходимо определить ее причину, т.е. определить оператор или фрагмент, содержащие ошибку. Причины ошибок могут быть как очевидны, так и очень глубоко скрыты. В целом сложность отладки обусловлена следующими причинами:  − требует от программиста глубоких знаний специфики управления используемыми техническими средствами, операционной системы, среды и языка программирования, реализуемых процессов, природы и специфики различных ошибок, методик отладки и соответствующих программных средств;  − психологически дискомфортна, так как необходимо искать собственные ошибки и, как правило, в условиях ограниченного времени;  − возможно взаимовлияние ошибок в разных частях программы, например, за счет затирания области памяти одного модуля другим из-за ошибок адресации;  − отсутствуют.  Большинство ошибок можно обнаружить по косвенным признакам посредством тщательного анализа текстов программ и результатов тестирования без получения дополнительной информации. При этом используют различные методы:  − ручного тестирования;  − индукции;  − дедукции;  − обратного прослеживания.  Метод ручного тестирования - самый простой и естественный способ данной группы. При обнаружении ошибки необходимо выполнить тестируемую программу вручную, используя тестовый набор, при работе с которыми была обнаружена ошибка.  Метод индукции Метод основан на тщательном анализе симптомов ошибки, которые могут проявляться как неверные результаты вычислений или как сообщение об ошибке. Если компьютер просто "зависает", то фрагмент проявления ошибки вычисляют, исходя из последних полученных результатов и действий пользователя.  По методу дедукции вначале формируют множество причин, которые могли бы вызвать данное проявление ошибки. Затем анализируя причины, исключают те, которые противоречат имеющимся данным. Если все причины исключены, то следует выполнить дополнительное тестирование исследуемого фрагмента. В противном случае наиболее вероятную гипотезу пытаются доказать. Если гипотеза объясняет полученные признаки ошибки, то ошибка найдена, иначе - проверяют следующую причину.  Отладка программ и обработка ошибок - это не одно и то же, но они тесно связаны друг с другом.  Отладка программ - это проверка и внесение исправлений в программу при ее разработке.  Обработка ошибок - это задание реакции на ошибки, которые возникают во время выполнения программы.  Все возможные ошибки можно разделить на три вида:  1 Ошибки компиляции. Возникают, если Java не может интерпретировать введенный текст, например, при использовании неправильного синтаксиса инструкции или задании неверного имени метода или свойства. Некоторые ошибки компиляции обнаруживаются при вводе инструкции, а другие - только перед выполнением программы. Данный тип ошибок обычно просто идентифицировать и исправить, поскольку Java выявляет их автоматически, а сами ошибки очевидны.  2 Ошибки выполнения. Возникают при выполнении программы, т.е. после успешной компиляции. Причиной таких ошибок может быть отсутствие данных или неправильная информация (например, данные, введенные пользователем). При этом выводится инструкция, при выполнении которой произошла ошибка. Ошибки данного типа тяжелее устранить: может понадобиться вывести значения переменных или свойств, а также другие данные, которые влияют на успешное выполнение программы.  3 Логические ошибки труднее всего заметить и устранить. Логические ошибки не приводят к прекращению компиляции или выполнения. Однако они являются причиной того, что программа не выдает желаемых результатов.  Компиляция — это процесс преобразования программы, написанной на алгоритмическом языке, в язык машинных кодов.  Точку останова (breakpoint) в Visual Studio можно помещать на любую строку кода, которая в действительности выполняется. Самый простой способ — щелчок на необходимой строке в окне редактора кода внутри затененной области вдоль левого края окна документа (или выделение нужной строки и нажатие клавиши <F9>). Это приводит к размещению в данной строке точки останова, которая вызывает прерывание процесса выполнения и передачу управления отладчику.  Слежения  После срабатывания точки останова обычно необходимо просмотреть значения переменных. Проще всего это сделать, наведя курсор мыши на имя интересующей переменной прямо в окне редактора кода.  Три предлагаемых в этом окне вкладки предназначены для наблюдения за переменными трех разных категорий:  − Вкладка Autos (Автоматические) позволяет просматривать значения нескольких последних переменных, к которым осуществлялся доступ в процессе выполнения программы.  − Вкладка Locals (Локальные) позволяет просматривать значения переменных, к которым получается доступ в методе, выполняемом в текущий момент  − Вкладка Watch (Слежение) позволяет просматривать значения любых интересующих переменных за счет явного указания их имен непосредственно в окне Watch.  Исключения  Исключения являются замечательным средством для обеспечения надлежащей обработки ошибок в поставляемом приложении. В случае правильного применения они позволяют обрести уверенность в том, что приложению удастся справиться с трудностями, а перед пользователем никогда не появится диалоговое окно с техническим описанием неполадки.  **Задание 2**  Изучить теоретический материал. Законспектировать основные определения.  Исключительная ситуация (или исключение) - это ошибка, которая возникает во время выполнения программы. Используя С# – подсистему обработки исключительных ситуаций, с такими ошибками можно справляться.  Ядром обработки исключений являются блоки try и catch. Эти ключевые слова работают "в одной связке";  **Задание 3**  Выполнить отладку кода разрабатываемой системы. Описать в дневнике-отчете обработанные исключительные ситуации в коде программы. Листинг кода с обработкой исключений вставить в отчет. «Приложение 17.3»  **Часть 2**  **Задание 1**  Проверил совместимость модулей программного средства. При выявлении ошибок исправить их. Сделал вывод о готовности Вашего ПС.  **Задание 2**  Листинг кода и скриншоты окон разработанного программного средства вставил в дневник-отчет. «Приложение 17.2» |  |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 25.05.2022,  26.05.2022 | **Практическая работа №25-26**  Тема: Тестирование программной системы.  Наименование работы: Формирование тестов. Проведение тестирования программной системы.  Цель: закрепление умений и навыков по верификации и валидации программного средства, тестирование программного средства.  воспитывающая – воспитание положительного отношения к знаниям, мотивов труда, интереса к предмету, внимательности, дисциплинированности, самостоятельности, инициативности.  развивающая – развивать самостоятельное мышление учащихся, умения пользоваться полученными знаниями.  **Задание 1**  Прочитайте и законспектируйте в дневник-отчет теоретический материал. (Определения верификации и тестирования; характеристики качества ПО, задачи верификации в рамках жизненного цикла ПО).  Верификация (verification) — это процесс оценки системы, чтобы понять, удовлетворяют ли результаты текущего этапа разработки условиям, которые были сформулированы в его начале.  Тестирование программного обеспечения (Software Testing) — проверка соответствия реальных и ожидаемых результатов поведения программы, проводимая на конечном наборе тестов, выбранном определённым образом.  С точки зрения ISO 9126, Качество (программных средств) можно определить, как совокупную характеристику исследуемого ПО с учётом следующих составляющих:  • Надёжность  • Сопровождаемость  • Практичность  • Эффективность  • Мобильность  • Функциональность  Задачи верификации в рамках жизненного цикла ПО  Все используемые на практике модели жизненного цикла по схеме организации работ являются разновидностями либо каскадной, либо итеративной модели, поэтому независимо от процесса разработки ПО верификации играет в нем ключевую роль, решая следующие задачи.   * Выявление дефектов (ошибок, недоработок, неполноты и пр.) различных артефактов разработки ПО (требований, проектных решений, документации или кода), что позволяет устранять их и поставлять пользователям и заказчикам более правильное и надежное ПО. * Выявление наиболее критичных и наиболее подверженных ошибкам частей создаваемой или сопровождаемой системы. * Контроль и оценка качества ПО во всех его аспектах. * Предоставление всем заинтересованным лицам (руководителям, заказчикам, пользователям и пр.) информации о текущем состоянии проекта и характеристиках его результатов. * Предоставление руководству проекта и разработчикам информации для планирования дальнейших работ, а также для принятия решений о продолжении проекта, его прекращении или передаче результатов заказчику.   **Задание 2**  Изучил и законспектировал методы тестирования: черного и белого ящика. Проанализировал и записал в дневник-отчет достоинств и недостатки каждого метода. Сделал вывод о применимости данных методов для тестирования ПП.  Тестирование «белого ящика» и «чёрного ящика»  При тестировании белого ящика (англ. white-box прозрачного ящика), разработчик теста имеет доступ к исходному коду программ и может писать код, который связан с библиотеками тестируемого ПО. Это типично для юнит-тестирования (англ. unit testing), при котором тестируются только отдельные части системы. Оно обеспечивает то, что компоненты конструкции — работоспособны и устойчивы, до определённой степени. При тестировании белого ящика используются метрики покрытия кода.  При тестировании чёрного ящика, тестировщик имеет доступ к ПО только через те же интерфейсы, что и заказчик или пользователь, либо через внешние интерфейсы, позволяющие другому компьютеру либо другому процессу подключиться к системе для тестирования. Как правило, тестирование чёрного ящика ведётся с использованием спецификаций или иных документов, описывающих требования к системе. Как правило, в данном виде тестирования критерий покрытия складывается из покрытия структуры входных данных, покрытия требований и покрытия модели (в тестировании на основе моделей).  Преимущества тестирования черного ящика включают в себя:   * Простота: облегчает тестирование проектов высокого уровня и сложных приложений * Экономия ресурсов: тестеры сосредоточены на функциональности программного обеспечения * Тестовые случаи: Сосредоточение внимания на функциональности программного обеспечения для облегчения быстрой разработки тестовых случаев * Обеспечивает гибкость: специальные знания программирования не требуются   Недостатки: - невозможно найти взаимоуничтожающиеся ошибки - некоторые ошибки возникают достаточно редко и потому их трудно найти и воспроизвести.  Достоинства метода белого ящика   * Оптимизация кода путем нахождения скрытых ошибок * Доступность структуры кода позволяет выбрать тип входных данных, необходимых для эффективного тестирования Возможность автоматизирования тест-кейсов   Недостатки белого ящика   * Поскольку знание кода и внутренней структуры является необходимым условием, для проведения такого тестирования требуется квалифицированный тестировщик, что увеличивает стоимость * И почти невозможно изучить каждый кусок кода, чтобы обнаружить скрытые ошибки, что может создать проблемы, приводящие к сбою приложения   **Задание 3**  Просмотрел презентацию Разработка тестов.ppt. Законспектировал в дневник-отчет основные моменты по разработке тестов. Просмотрел файл Пример разработка тестов.xls. Используя Шаблон для разработки тестов. xls, составила тестовые сценарии функционального тестирования для своего приложения и протестируйте его. «Приложение 18.1» Сделал вывод.  Дефект – это несоответствие требованиям или функциональным спецификациям.  Отчёт об ошибке – это технический документ, написанный с целью:  – предоставить информацию о проблеме, ей свойствах и  последствиях;  – приоритизировать устранения;  – помочь программистам обнаружить и и устранить источник проблемы.  Основная цель написания отчёта об ошибке – устранение  ошибки.  Жизненный цикл дефекта  Обнаружен (submitted). Итак, тестировщик находит дефект и представляет его на рассмотрение в систему управления дефектами. С этого момента баг начинает свою официальную жизнь и о его существовании знают необходимые люди.  Назначен (assigned). Далее ведущий разработчик рассматривает дефект и назначает его исправление кому-то из команды разработчиков.  Исправлен (fixed). Разработчик, которому было назначено исправление дефекта, исправляет его и сообщает о том, что задание выполнено.  Проверен (verified). Тестировщик, который обнаружил ошибку проверяет на новом билде (в котором исправление данной ошибки заявлено), исправлен ли дефект на самом деле. И  только в том случае, если ошибка не проявится на новом билде, тестировщик меняет статус бага на Verified.  Открыт заново (reopened). Если баг проявляется на новом билде, тестировщик снова открывает этот дефект. Баг приобретает статус Reopened.  Отклонён (declined). Баг может быть отклонён. Во-первых, потому, что для заказчика какие-то ошибки перестают быть актуальными. Во-вторых, это может случится по вине тестировщика из-за плохого знания продукта, требований (дефекта на самом деле нет).  Отложен (deferred). Если исправление конкретного бага сейчас не очень важно или заказчик пока думает, или мы ждём какую-то информацию, от которой зависит исправление бага, тогда баг приобретает статус Deferred.  Закрытые (closed) баги. Закрытым считается баг в состояниях Проверен (verified) и Отклонён (declined).  Открытые (open) баги. Открытыми являются баги в остояниях Обнаружен (submitted), Назначен (assigned), Открыт заново (reopened). Иногда к открытым относят и баги в состояниях Исправлен (fixed) и Отложен (deferred).  На данной стадии разработки проект работоспособен.  Продолжение разработки продолжится позже.  **Контрольные вопросы:**   1. Верификация (verification) — это процесс оценки системы, чтобы понять, удовлетворяют ли результаты текущего этапа разработки условиям, которые были сформулированы в его начале. 2. Факторы и атрибуты относятся к характеристикам качества программного продукта: Надёжность, Сопровождаемость, Практичность, Эффективность, Мобильность, Функциональность. 3. Тестирование программного обеспечения (Software Testing) — проверка соответствия реальных и ожидаемых результатов поведения программы, проводимая на конечном наборе тестов, выбранном определённым образом. Баг — это отклонение фактического результата от ожидаемого результата. 4. Основные методы тестирования ПО: Модульное тестирование, интеграционное тестирование, системное тестирование, альфа-тестирование, бета-тестирование. |  |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 27.05.2022,  28.05.2022 | **Практическая работа №27-28**  Тема: Документирование программного продукта  Наименование работы: Разработка структуры электронного варианта документации. Оформление документации в соответствие с нормативными требованиями.  Цель: закрепление умений и навыков по документации программного средства, тестирование программного средства.  воспитывающая – воспитание положительного отношения к знаниям, мотивов труда, интереса к предмету, внимательности, дисциплинированности, самостоятельности, инициативности.  развивающая – развивать самостоятельное мышление учащихся, умения пользоваться полученными знаниями.  **Задание 1**  Прочитал методические указания. Составил краткий конспект по теме «Документация ПО» в дневник-отчет.  Документация на программное обеспечение — это документы, сопровождающие некоторое программное обеспечение (ПО) — программу или программный продукт. Эти документы описывают то, как работает программа и/или то, как её использовать.  Документирование — это важная часть в разработке программного обеспечения, но часто ей уделяется недостаточно внимания.  Существует четыре основных типа документации на ПО:   1. архитектурная/проектная — обзор программного обеспечения, включающий описание рабочей среды и принципов, которые должны быть использованы при создании ПО 2. техническая — документация на код, алгоритмы, интерфейсы, API 3. пользовательская — руководства для конечныхпользователей, администраторов системы и другого персонала 4. маркетинговая   Архитектурная/проектная документация  Проектная документация обычно описывает продукт в общих чертах. Не описывая того, как что-либо будет использоваться, она скорее отвечает на вопрос «почему именно так?» Описываются причины, почему какой-либо класс сконструирован определённым образом, выделяются паттерны, в некоторых случаях даже даются идеи, как можно будет выполнить улучшения в дальнейшем. Ничего из этого не входит в техническую или пользовательскую документацию, но всё это действительно важно для проекта.  Техническая документация  Это именно то, что подразумевают под термином документация большинство программистов. При создании программы, одного лишь кода, как правило, недостаточно. Должен быть предоставлен некоторый текст, описывающий различные аспекты того, что именно делает код. Такая документация часто включается непосредственно в исходный код или предоставляется вместе с ним.  Пользовательская документация  В отличие от технической документации, сфокусированной на коде и том, как он работает, пользовательская документация описывает лишь то, как использовать программу. В случае если продуктом является программная библиотека, пользовательская документация и документация на код становятся очень близкими, почти эквивалентными понятиями. Но в общем случае, это не так. Обычно, пользовательская документация представляет собой руководство пользователя, которое описывает каждую функцию программы, а также шаги, которые нужно выполнить для использования этой функции.  Маркетинговая документация  Такая форма документации имеет целью:   1. подогреть интерес к продукту у потенциальных пользователей 2. информировать их о том, что именно делает продукт, с тем чтобы их ожидания совпадали с тем что они получат объяснить положение продукта по сравнению с конкурирующими решениями 3. Одна из хороших маркетинговых практик — предоставление слогана — простой запоминающейся фразы, иллюстрирующей то что мы хотим донести до пользователя, а также характеризующей ощущение, которое создаёт продукт.   Часто бывает так, что коробка продукта и другие маркетинговые материалы дают более ясную картину о возможностях и способах использования программы, чем всё остальное.  **Задание 2**  Для ПС разработал проектную, техническую и пользовательскую документацию. Откорректировал техническое задание.  **Задание 3**  Для ПС разработала руководство пользователя и руководство программиста. «Приложение19.1»  **Контрольные вопросы:**   1. Документация на программное обеспечение — это документы, сопровождающие некоторое программное обеспечение (ПО) — программу или программный продукт. 2. Типы документации на ПО:  * архитектурная/проектная — обзор программного обеспечения, включающий описание рабочей среды и принципов, которые должны быть использованы при создании ПО * техническая — документация на код, алгоритмы, интерфейсы, API * пользовательская — руководства для конечныхпользователей, администраторов системы и другого персонала * маркетинговая  1. К проектной документации относятся:  * архитектурные решения * схемы планировочной организации земельного участка * конструктивные и объемно-планировочные решения * проекты организации строительства, работ по сносу или демонтажу объектов капитального строительства * перечни мероприятий по охране окружающей среды, обеспечению пожарной безопасности, обеспечению доступа инвалидов * сметы на строительство объектов  1. Пользовательская документация описывает лишь то, как использовать программу. 2. Должен быть предоставлен некоторый текст, описывающий различные аспекты того, что именно делает код. 3. Руководство пользователя содержит следующие разделы:  * общие сведения о программном продукте * описание установки * описание запуска * инструкции по работе (или описание пользовательского интерфейса) * сообщения пользователю  1. Она включает следующие  * Внешнее описание программного средства. * Описание архитектуры программного средства, включая внешнюю спецификацию каждой ее программы. * Для каждой программы программного средства - описание ее модульной * структуры, включая внешнюю спецификацию каждого включенного в нее * Для каждого модуля - его спецификация и описание его строения * Тексты модулей на выбранном языке программирования * Документы установления достоверности программного средства, описывающие, как устанавливалась достоверность каждой программы программного средства и как информация об установлении достоверности связывалась с требованиями к программному средству. |  |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 30.05.2022 | **Практическая работа №29**  Тема: Сопровождение программного продукта.  Наименование работы: Организация сопровождения программного продукта. Управление требованиями.  Цель: закрепление умений и навыков организации сопровождения ПО, обучения пользователей.  **Задание 1**  Изучил теоретический материал практической работы, законспектировал основные определения.  Сопровождение программного обеспечения — процесс улучшения, оптимизации и устранения дефектов программного обеспечения (ПО) после передачи в эксплуатацию.  Сопровождение ПО — это одна из фаз жизненного цикла программного обеспечения, следующая за фазой передачи ПО в эксплуатацию. В ходе сопровождения в программу вносятся изменения, с тем, чтобы исправить обнаруженные в процессе использования дефекты и недоработки, а также для добавления новой функциональности, с целью повысить удобство использования и применимость ПО.  Успех внедрения информационных систем зависит от многих факторов, из которых наиболее важными являются создание и обновление документации о системе, а также обучение и поддержка пользователей.  25-30% успеха проекта внедрения информационной системы зависит именно от лояльности конечных пользователей: их готовности к использованию продукта и их уверенности в том, что в любой момент они смогут получить помощь оперативно и в нужном объеме.  Быстрым и эффективным обучением, материалами, а также поддержкой их работы в информационной системе на протяжении всего процесса освоения и работы системы.  **Задание 2**  Продумал и законспектировал основные моменты, которые необходимо рассмотреть при обучении конечных пользователей.  Для работы с приложением необходимо показать все переходы по вкладке меню, обязательно рассказать про функции добавления и редактирования, и показать, как приложение и при каком условии удаляет данные.  **Задание 3**  Создал презентацию с описанием вашего ПП. |  |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 31.05.2022 | **Практическая работа №30**  Тема: Сопровождение программного продукта.  Наименование работы: Выявление, оценка и формализация изменения требований к программному продукту.  Цель: изучить методы проведения верификации и валидации программных продуктов, научиться выявлять необходимые для изменения требования, определять необходимые изменения.  **Задание 1**  Изучил статью «Верификация и валидация», законспектировал.  Термин "верификация" (verification) в русскоязычной литературе обычно переводят, как "проверка". Термин "валидация" - как "проверка правильности", "аттестация", "утверждение".  Согласно стандарту IЕЕЕ 1012-1986, верификация представляет собой процесс оценивания системы или компонента с целью определить, удовлетворяют ли результаты некой фазы условиям, наложенным в начале данной фазы. Валидация в этом же стандарте определяется, как процесс оценивания системы или компонента во время или по окончании процесса разработки с целью определить, удовлетворяет ли она указанным требованиям.  Двусмысленность (несоответствие свойству ясности, закладывает под проект "бомбу замедленного действия". На практике требование, сформулированное двусмысленным образом, может привести к различным его интерпретациям представителями Разработчика и Заказчика.  Под "золочением" понимают такие ситуации, когда разработчики добавляют функции, которых нет в спецификации, но им кажется, что это понравится пользователям. Зачастую же клиентам не нужны такие избыточные возможности, получается, что время, отведенное на реализацию, тратится впустую. Эта ситуация возникает в случае, когда, во-первых, в коллективе Разработчика присутствуют творческие личности (ведь далеко не всякая команда станет проявлять инициативу и делать сверх того, о чем ее просили), во вторых - существует разрыв в прохождении информации от Заказчика к Разработчику. Инициативный разработчик "золотит" продукт из самых лучших побуждений, но, возможно, он плохо знаком с бизнес-процессом Заказчика и заложенные им "фичи" попросту не будут востребованы.  Минимальная спецификация уместна, если имеет место наличие одновременно трех обстоятельств: а) цена контракта и размеры проекта таковы, что разработка развернутого ТЗ экономически нецелесообразна; б) коллектив Разработчика обладает достаточной степенью профессионализма и опыта выполнения проектов в смежных областях, чтобы уметь создавать по краткой спецификации продукт, который пройдет приемку Заказчиком; в) между Заказчиком и Разработчиком существуют конструктивные отношения и обе стороны понимают и принимают риски мини-спецификации.  Методы и средства проверки требований  Они разнятся по ряду параметров. Так, различают: по широте анализа - просмотр (выборочная проверка) и сквозной контроль (тотальная проверка); по степени формализации - неофициальные процедуры, процедуры, проводимые по формальным правилам (инспекции, экспертизы); по составу группы проверки - с (без) участием автора, с (без) участием менеджера проекта, с (без) участием представителей внешних организаций; по используемым средствам - тексты требований, тестовые сценарии, критерии приемлемости, прототипы.  Неофициальные просмотры требований  Различают несколько способов неофициальных просмотров требований: просмотр "за столом", коллективная проверка, критический анализ.  В группу инспекции входят лидер, регистратор, рецензент и несколько (от 2 до 5) инспекторов. Члены команды инспектирования могут специализироваться в различных областях экспертизы (обладать различными областями компетенции), например, предметной области, методах проектирования, языке и т.п. В заданный момент (промежуток) времени инспекции проводятся в отношении отдельного небольшого фрагмента продукта (в большинстве случаев, фокусируясь на отдельных функциональных или других характеристиках; часто, отталкиваясь от отдельных бизнес-правил, функциональных требований или атрибутов качества, прим. автора).  Решение о завершении инспекции принимается в соответствии с одним (любым) из трех критериев:  1. Принятие с отсутствием либо малой необходимостью переработки  2. Принятие с проверкой переработанных фрагментов  3. Необходимость повторной инспекции.  **Задание 2**  Выполнил верификацию разработанного программного средства.   1. Разработать понятный интерфейс для удобства пользования сотрудников учреждений образования 2. Реализовать несколько уровней доступа к информации: Методист, учитель физической культуры. 3. Показать пользователю информацию необходимую для организации работы: учреждения образования, учащиеся, методические объединения, преподаватели по физической культуре, методические рекомендации, виды спорта, уровни соревнований, информация о проведении соревнований. 4. Создать возможность добавления, редактирования, удаления информации.   Выше перечисленные пункты соответствуют требованиям.  **Задание 3**  Определил перспективу развития разработанного программного средства, проанализировать требования к ПО в ТЗ, выявить какие требования требуют изменения/дополнения в соответствии с определенной перспективой развития.  Работу над проектом планирую продолжать:   * Разработать отдельный интерфейс для пользователей (учителей физкультуры) * Разработать отдельное окто администратора с полным функционалам * Ограничить доступ по уровням пользователя |  |

|  |  |  |
| --- | --- | --- |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 01.06.2022 | **Практическая работа №31**  Тема: Сопровождение программного продукта.  Наименование работы: Редактирование функциональной и бизнес моделей в соответствии с внесенными в ТЗ изменениями. Редактирование моделей “сущность-связь” и потоков данных в соответствии с внесенными в ТЗ изменениями.  Цель: выполнить редактирование функциональной и бизнес моделей, моделей «сущность-связь» и потоков данных в соответствии с измененными требованиями в Практической работе 30  **Задание 1**  Просмотрев функциональную и бизнес модели в соответствии с измененным ТЗ я сделал вывод, что модели соответствуют ТЗ. Они представлены в приложении «Приложение 4.1».  **Задание 2**  Просмотрев модели “сущность-связь” и потоков данных в соответствии ТЗ я сделал вывод, что модели соответствуют ТЗ. Они представлены в приложении «Приложение 6.1», «Приложение 5.1» |  |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 02.06.2022 | **Практическая работа №32**  Тема: Сопровождение программного продукта.  Наименование работы: Редактирование UML диаграмм в соответствии с внесенными в ТЗ изменениями.  Цель: выполнить редактирование UML диаграмм в соответствии с измененными требованиями в Практической работе 30.  **Задание 1**  Просмотрев диаграммы прецедентов в соответствии с ТЗ, я сделал вывод, что диаграмма соответствует ТЗ. Она представлены в «Приложении 11.1».  **Задание 2**  Просмотрев диаграммы деятельности в соответствии с ТЗ, я сделал вывод, что диаграмма соответствует ТЗ. Она представлены в «Приложении 12.1».  **Задание 3**  Просмотрев диаграммы последовательности в соответствии с ТЗ, я сделал вывод, что диаграмма соответствует ТЗ. Она представлена в «Приложении 13.1».  **Задание 4**  Просмотрев диаграммы классов в соответствии с ТЗ, я сделал вывод, что диаграмма соответствует ТЗ. Она представлена в «Приложении 14.1». |  |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 03.06.2022 | **Практическая работа №33**  Тема: Сопровождение программного продукта.  Наименование работы: Внесение изменений в программный продукт.  Цель: закрепление умений и навыков по кодированию модулей программного средства, корректировке кода готового ПС.  воспитывающая – воспитание положительного отношения к знаниям, мотивов труда, интереса к предмету, внимательности, дисциплинированности, самостоятельности, инициативности.  развивающая – развивать самостоятельное мышление учащихся, умения пользоваться полученными знаниями.  **Задание 1**  Просмотрев требования, сделал вывод что в программе не чего не изменилось из пользовательского интерфейса  **Задание 2**  Проверил совместимость модулей моего программного средства. Выявлении ошибок не обнаружено. На данной стадии разработки проект работоспособен. Продолжение разработки продолжится позже. |  |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 04.06.2022 | **Практическая работа №34**  Тема: Сопровождение программного продукта.  Наименование работы: Тестирование ПО.  Цель: закрепление умений и навыков по верификации и валидации программного средства, тестирование программного средства.  воспитывающая – воспитание положительного отношения к знаниям, мотивов труда, интереса к предмету, внимательности, дисциплинированности, самостоятельности, инициативности.  развивающая – развивать самостоятельное мышление учащихся, умения пользоваться полученными знаниями.  **Задание 1**  Просмотрев свое приложение и составленный тестовый сценарий функционального тестирования, я сделал вывод, что все соответствует первоначальному приложению. Мое приложение работоспособно и прошло все тесты.  **Контрольные вопросы:**   1. Верификация – это процесс определения, выполняют ли программные средства и их компоненты требования, наложенные на них в последовательных этапах жизненного цикла разрабатываемой программной системы. 2. Факторы и атрибуты относятся к характеристикам качества программного продукта: Надёжность, Сопровождаемость, Практичность, Эффективность, Мобильность, Функциональность. 3. Тестирование программного обеспечения (Software Testing) — проверка соответствия реальных и ожидаемых результатов поведения программы, проводимая на конечном наборе тестов, выбранном определённым образом. 4. Баг — это отклонение фактического результата от ожидаемого результата. 5. Основные методы тестирования ПО: Модульное тестирование, интеграционное тестирование, системное тестирование, альфа-тестирование, бета-тестирование. |  |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 06.06.2022 | **Практическая работа №35**  Тема: Сопровождение программного продукта  Наименование работы:Разработка комплекта документации в соответствии с нормативными требованиями и методическими указаниями.  Цель: закрепление умений и навыков по документации программного средства, тестирование программного средства.  воспитывающая – воспитание положительного отношения к знаниям, мотивов труда, интереса к предмету, внимательности, дисциплинированности, самостоятельности, инициативности.  развивающая – развивать самостоятельное мышление учащихся, умения пользоваться полученными знаниями.  **Задание 1**  Просмотрев свое ПС и руководство пользователя и руководство программиста, я сделал вывод, что все соответствует руководству пользователя и руководству программиста. Оно представлено в «Приложении 19.1»  **Контрольные вопросы:**   1. Документация на программное обеспечение — это документы, сопровождающие некоторое программное обеспечение (ПО) — программу или программный продукт. 2. Типы документации на ПО:  * архитектурная/проектная — обзор программного обеспечения, включающий описание рабочей среды и принципов, которые должны быть использованы при создании ПО * техническая — документация на код, алгоритмы, интерфейсы, API * пользовательская — руководства для конечныхпользователей, администраторов системы и другого персонала * маркетинговая  1. К проектной документации относятся:  * архитектурные решения * схемы планировочной организации земельного участка * конструктивные и объемно-планировочные решения * проекты организации строительства, работ по сносу или демонтажу объектов капитального строительства * перечни мероприятий по охране окружающей среды, обеспечению пожарной безопасности, обеспечению доступа инвалидов * сметы на строительство объектов  1. Пользовательская документация описывает лишь то, как использовать программу. 2. Должен быть предоставлен некоторый текст, описывающий различные аспекты того, что именно делает код. 3. Руководство пользователя содержит следующие разделы:  * общие сведения о программном продукте * описание установки * описание запуска * инструкции по работе (или описание пользовательского интерфейса) * сообщения пользователю  1. Она включает следующие  * Внешнее описание программного средства. * Описание архитектуры программного средства, включая внешнюю спецификацию каждой ее программы. * Для каждой программы программного средства - описание ее модульной * структуры, включая внешнюю спецификацию каждого включенного в нее * Для каждого модуля - его спецификация и описание его строения * Тексты модулей на выбранном языке программирования * Документы установления достоверности программного средства, описывающие, как устанавливалась достоверность каждой программы программного средства и как информация об установлении достоверности связывалась с требованиями к программному средству. |  |
| Дата | Содержание выполняемой работы | Подпись  руководителя |
| 07.06.2022 | **Практическая работа №36**  Тема: Сопровождение программного продукта.  Наименование работы: Защита результатов практики.  Цель: анализ умений и навыков по разработке программного обеспечения.  воспитывающая – воспитание положительного отношения к знаниям, мотивов труда, интереса к предмету, внимательности, дисциплинированности, самостоятельности, инициативности.  развивающая – развивать самостоятельное мышление учащихся, умения пользоваться полученными знаниями.  **Задание 1**  Составил отчет. Кратко проанализировал основные моменты прохождения практики:  За время прохождения учебной практики по разработке и сопровождению программного обеспечения было выполнено 36 практических работ с общим объемом часов – 216. Каждая практическая работа имела как теоретическую часть, так и индивидуальные задания. Для выполнения индивидуальных заданий использовалось следующее программное обеспечение: Visual Studio 2022, Visual Studio Code, ErWin Process Modeler, Draw.io, phpMyAdmin, Excel, MySQL, Notepad++, Power. Дневник-отчет создавался в приложении Microsoft Word 2016. Общий объем дневника отчета составляет 75 страницы. Все материалы записаны на компакт-диск и приложены к дневнику-отчету.  **Задание 2**  Продемонстрировала разработанное программное средство. |  |