АВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ОРГАНИЗАЦИЯ

«ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ»

«ОТКРЫТЫЙ ТАВРИЧЕСКИЙ КОЛЛЕДЖ»

УТВЕРЖДАЮ

Директор

АНО «ПОО» «Открытый

Таврический колледж»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Г.П. Узунова

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_20\_\_ г.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ

ПРАКТИЧЕСКИХ ЗАНЯТИЙ

ПО междисциплинарному курсу

**МДК.02.02 Инструментальные средства разработки программного обеспечения**

ПО СПЕЦИАЛЬНОСТИ

09.02.07 Информационные системы и программирование

Квалификация: программист

БАЗОВЫЙ УРОВЕНЬ ПОДГОТОВКИ

ФОРМА ОБУЧЕНИЯ

ОЧНАЯ

Симферополь, 2025 г.

**МДК.02.02 Инструментальные средства разработки программного обеспечения**

**Методические указания к практическому занятию № 1**

**Тема:** Разработка структуры проекта. Разработка модульной структуры проекта (диаграммы модулей).

**Количество часов**: 2

**Цель**:

- *обучающая*: рассмотреть особенности разработки структуры проекта; сформировать практические умения и навыки разработки модульной структуры проекта;

- *воспитательная*: выработать умение мыслить, научить логически мыслить; оценить степень работоспособности; развивать познавательные возможности, внимание; содействовать развитию профессиональных качеств;

- *развивающая*: развивать умения и навыки применять: теорию при решении задач, навыки самостоятельной работы с методическими указаниями к практическому занятию, осуществлять самоконтроль, язык терминов.

**Задания:**

1. Изучить материал представленный в теоретической части (1 балл).

2. Сформировать команду проекта в составе двух человек, распределить роли. Выполнить задания практической части по варианту. (4 балла).

3. Защитить выполненное задание (1 балл).

*Выводы:* выполнение практического занятия способствует формированию практических навыков по разработке структуры проекта и разработка модульной структуры проекта.

**1. Теоретическая часть**

**Общие сведения о разработке программного обеспечения**

Проблемы управления программными проектами впервые проявились в 60-х – начале 70-х годов, когда провалились многие большие проекты по разработке программных продуктов. Были зафиксированы задержки в создании ПО, оно было ненадежным, затраты на разработку в несколько раз превосходили первоначальные оценки, созданные программные системы часто имели низкие показатели производительности. Причины провалов коренились в тех подходах, которые использовались в управлении проектами. Применяемая методика была основана на опыте управления техническими проектами и оказалась неэффективной при разработке программного обеспечения.

Важно понимать разницу между профессиональной разработкой ПО и любительским программированием. Необходимость управления программными проектами вытекает из того факта, что процесс создания профессионального ПО всегда является субъектом бюджетной политики организации, где оно разрабатывается, и имеет временные ограничения. Работа *руководителя программного проекта* по большому счету заключается в том, чтобы гарантировать выполнение этих бюджетных и временных ограничений с учетом бизнес-целей организации относительно разрабатываемого ПО.

*Руководители проектов* призваны спланировать все этапы разработки программного продукта. Они также должны контролировать ход выполнения работ и соблюдения всех требуемых стандартов.

Постоянный контроль за ходом выполнения работ необходим для того, чтобы процесс разработки не выходил за временные и бюджетные ограничения. Хорошее управление не гарантирует успешного завершения проекта, но плохое управление обязательно приведет к его провалу. Это может выразиться в задержке сроков сдачи готового ПО, в превышении сметной стоимости проекта и в несоответствии готового ПО спецификации требований.

Процесс разработки ПО существенно отличается от процессов реализации технических проектов, что порождает определенные сложности в управлении программными проектами:

1.    *Программный продукт нематериален.*Программное обеспечение нематериально, его нельзя увидеть или потрогать. Руководитель программного проекта не видит процесс "роста" разрабатываемого ПО. Он может полагаться только на документацию, которая фиксирует процесс разработки программного продукта.

2.       *Не существует стандартных процессов разработки ПО.*Насегодняшний день не существует четкой зависимости между процессом создания ПО и типом создаваемого программного продукта. Другие технические дисциплины имеют длительную историю, процессы разработки технических изделий многократно опробованы и проверены. Процессы создания большинства технических систем хорошо изучены. Изучением же процессов создания ПО специалисты занимаются только последнее время. Поэтому пока нельзя точно предсказать, на каком этапе процесса разработки ПО могут возникнуть проблемы, угрожающие всему программному проекту.

3.    *Большие программные проекты - это часто "одноразовые" проекты.*Большие программные проекты,как правило,значительно отличаются от проектов, реализованных ранее. Поэтому, чтобы уменьшить неопределенность в планировании проекта, руководители проектов должны обладать очень большим практическим опытом. Но постоянные технологические изменения в компьютерной технике и коммуникационном оборудовании обесценивают предыдущий опыт. Знания и навыки, накопленные опытом, могут не востребоваться в новом проекте.

Перечисленные отличия могут привести к тому, что реализация проекта выйдет из временного графика или превысит бюджетные ассигнования. Программные системы зачастую оказываются новинками как в "идеологическом", так и в техническом плане. Поэтому, предвидя возможные проблемы в реализации программного проекта, следует всегда помнить, что многим из них свойственно выходить за рамки временных и бюджетных ограничений.

**Процесс управления разработкой программного обеспечения**

Невозможно описать и стандартизировать все работы, выполняемые в проекте по созданию ПО. Эти работы весьма существенно зависят от организации, где выполняется разработка ПО, и от типа создаваемого программного продукта. Но всегда можно выделить следующие:

1. Написание предложений по созданию ПО.
2. Планирование и составление графика работ по созданию ПО.
3. Оценивание стоимости проекта.
4. Подбор персонала.
5. Контроль за ходом выполнения работ.
6. Написание отчетов и представлений.

Первая стадия программного проекта может состоять из написания предложений по реализации этого проекта. Предложения должны содержать описание целей проектов и способов их достижения. Они также обычно включают в себя оценки финансовых и временных затрат на выполнение проекта. При необходимости здесь могут приводиться обоснования для передачи проекта на выполнение сторонней организации или команде разработчиков.

*Написание предложений*—очень ответственная работа,таккак для многих организаций вопрос о том, будет ли проект выполняться самой организацией или разрабатываться по контракту сторонней компанией, является критическим. Не существует каких-либо рекомендаций по написанию предложений, многое здесь зависит от опыт.

На этапе *планирования проекта* определяются процессы, этапы и полученные на каждом из них результаты, которые должны привести к выполнению проекта. Реализация этого плана приведет к достижению целей проекта. Определение стоимости проекта напрямую связано с его планированием, поскольку здесь оцениваются ресурсы, требующиеся для выполнения плана.

*Контроль за ходом выполнения работ (мониторинг проекта)*— это непрерывный процесс, продолжающийся в течение всего срока реализации проекта. Руководитель должен постоянно отслеживать ход реализации проекта и сравнивать фактические и плановые показатели выполнения работ с их стоимостью. Хотя многие организации имеют механизмы формального мониторинга работ, опытный руководитель может составить ясную картину о стадии развитии проекта просто путем неформального общения с разработчиками.

Неформальный мониторинг часто помогает обнаружить потенциальные проблемы, которые в явном виде могут обнаружиться позднее. Например, ежедневное обсуждение хода выполнения работ может выявить отдельные недоработки в создаваемом программном продукте. Вместо ожидания отчетов, в которых будет отражен факт "пробуксовки" графика работ, можно обсудить со специалистами намечающиеся программистские проблемы и не допустить срыва графика работ.

В   течение реализации проекта обычно происходит несколько формальных контрольных проверок хода выполнения работ по созданию ПО. Такие проверки должны дать общую картину хода реализации проекта в целом и показать, насколько уже разработанная часть ПО соответствует целям проекта.

Время выполнения больших программных проектов может занимать несколько лет. В течение этого времени цели и намерения организации, заказавшей программный проект, могут существенно измениться. Может оказаться, что разрабатываемый программный продукт стал уже ненужным либо исходные требования к создаваемому ПО просто устарели и их необходимо кардинально менять. В такой ситуации руководство организации-разработчика может принять решение о прекращении разработки ПО или об изменении проекта в целом с тем, чтобы учесть изменившиеся цели и намерения организации-заказчика.

Руководители проектов обычно обязаны сами *подбирать исполнителей*для своих проектов.В идеальном случае профессиональный уровень исполнителей должен соответствовать той работе, которую они будут выполнять в ходе реализации проекта. Однако во многих случаях руководители должны полагаться на команду разработчиков, которая далека от идеальной. Такая ситуация может быть вызвана следующими причинами:

1.       Бюджет проекта не позволяет привлечь высококвалифицированный персонал. В таком случае за меньшую плату привлекаются менее квалифицированные специалисты.

2.       Бывают ситуации, когда невозможно найти специалистов необходимой квалификации как в самой организации-разработчике, так и вне ее. Например, в организации "лучшие люди" могут быть уже заняты в других проектах.

3.    Организация хочет повысить профессиональный уровень своих работников. В этом случае она может привлечь к участию в проекте неопытных или недостаточно квалифицированных работников, чтобы они приобрели необходимый опыт и поучились у более опытных специалистов.

Таким образом, почти всегда подбор специалистов для выполнения проекта имеет определенные ограничения и не является свободным. Вместе с тем необходимо, чтобы хотя бы несколько членов группы разработчиков имели квалификацию и опыт, достаточные для работы над данным проектом. В противном случае невозможно избежать ошибок в разработке ПО.

Руководитель проекта обычно обязан посылать *отчеты* о ходе его выполнения как заказчику, так и подрядным организациям. Это должны быть краткие документы, основанные на информации, извлекаемой из подробных' отчетов о проекте. В этих отчетах должна быть та информация, которая позволяет четко оценить степень готовности создаваемого программного продукта.

В **рамках курса выделены следующие роли в группе по разработке ПО:**

1. **Руководитель – общее руководство проектом, написание документации, общение с заказчиком ПО.**
2. **Системный аналитик – разработка требований (составление технического задания, проекта программного обеспечения)**
3. **Тестер – составление плана тестирования и аттестации готового ПО (продукта), составление сценария тестирования, базовый пример, проведение мероприятий по плану тестирования.**
4. **Разработчик – моделирование компонент программного обеспечения, кодирование.**

**Планирование проекта разработки программного обеспечения**

Эффективное управление программным проектом напрямую зависит от правильного планирования работ, необходимых для его выполнения. План помогает руководителю предвидеть проблемы, которые могут возникнуть на каких-либо этапах создания ПО, и разработать превентивные меры для их предупреждения или решения. План, разработанный на начальном этапе проекта, рассматривается всеми его участниками как руководящий документ, выполнение которого должно привести к успешному завершению проекта. Этот первоначальный план должен максимально подробно описывать все этапы реализации проекта.

Процесс планирования начинается, исходя из описания системы, с определения проектных ограничений (временные ограничения, возможности наличного персонала, бюджетные ограничения и т.д.). Эти ограничения должны определяться параллельно с оцениванием проектных параметров, таких как структура и размер проекта, а также распределением функций среди исполнителей. Затем определяются этапы разработки и то, какие результаты документация, прототипы, подсистемы или версии программного продукта) должны быть получены по окончании этих этапов. Далее начинается циклическая часть планирования. Сначала разрабатывается график работ по выполнению проекта или дается разрешение на продолжение использования ранее созданного графика. После этого проводится контроль выполнения работ и отмечаются расхождения между реальным и плановым ходом работ.

Далее, по мере поступления новой информации о ходе выполнения проекта, возможен пересмотр первоначальных оценок параметров проекта. Это, в свою очередь, может привести к изменению графика работ. Если в результате этих изменений нарушаются сроки завершения проекта, должны быть пересмотрены (и согласованы с заказчиком ПО) проектные ограничения.

Конечно, большинство руководителей проектов не думают, что реализация их проектов пройдет гладко, без всяких проблем. Желательно описать возможные проблемы еще до того, как они проявят себя в ходе выполнения проекта. Поэтому лучше составлять "пессимистические" графики работ, чем "оптимистические". Но, конечно, невозможно построить план, учитывающий все, в том числе случайные, проблемы и задержки выполнения проекта, поэтому и возникает необходимость периодического пересмотра проектных ограничений и этапов создания программного продукта.

План проекта должен четко показать ресурсы, необходимые для реализации проекта, разделение работ на этапы и временной график выполнения этих этапов. В некоторых организациях план проекта составляется как единый документ, содержащий все виды планов, описанных выше. В других случаях план проекта описывает только технологический процесс создания ПО. В таком плане обязательно присутствуют ссылки на планы других видов, но они разрабатываются отдельно от плана проекта.

Детализация планов проектов очень разнится в зависимости от типа разрабатываемого программного продукта и организации-разработчика. Но в любом случае большинство планов содержат следующие разделы.

1.       *Введение*.Краткое описание целей проекта и проектных ограничений (бюджетных, временных и т.д.), которые важны для управления проектом.

2.       *Организация выполнения проекта*.Описание способа подборакоманды разработчиков и распределение обязанностей между членами команды.

3.       *Анализ рисков*.Описание возможных проектных рисков,вероятности их проявления и стратегий, направленных на их уменьшение.

4.    *Аппаратные и программные ресурсы, необходимые для реализации проекта*.Перечень аппаратных средств и программногообеспечения, необходимого для разработки программного продукта. Если аппаратные средства требуется закупать, приводится их стоимость совместно с графиком закупки и поставки.

5.       *Разбиение работ на этапы*.Процесс реализации проекта разбивается на отдельные процессы, определяются этапы выполнения проекта, приводится описание результатов ("выходов") каждого этапа и контрольные отметки.

6.       *График работ*.В этом графике отображаются зависимостимежду отдельными процессами (этапами) разработки ПО, оценки времени их выполнения и распределение членов команды разработчиков по отдельным этапам.

7.    *Механизмы мониторинга и контроля за ходом выполнения проекта*.Описываются предоставляемые руководителем отчеты о ходе выполнения работ, сроки их предоставления, а также механизмы мониторинга всего проекта.

План должен регулярно пересматриваться в процессе реализации проекта. Одни части плана, например график работ, изменяются часто, другие более стабильны. Для внесения изменений в план требуется специальная организация документопотока, позволяющая отслеживать эти изменения.

**Общие сведения о требованиях к информационным системам**

Проблемы, которые приходится решать специалистам в процессе создания программного обеспечения, очень сложны. Природа этих проблем не всегда ясна, особенно если разрабатываемая программная система инновационная. В частности, трудно чѐтко описать те действия, которые должна выполнять система. Описание функциональных возможностей и ограничений, накладываемых на систему, называется требованиями к этой системе, а сам процесс формирования, анализа, документирования и проверки этих функциональных возможностей и ограничений – разработкой требований.

Требования подразделяются на пользовательские и системные. Пользовательские требования – это описание на естественном языке (плюс поясняющие диаграммы) функций, выполняемых системой, и ограничений, накладываемых на неѐ. Системные требования – это описание особенностей системы (архитектура системы, требования к параметрам оборудования и т.д.), необходимых для эффективной реализации требований пользователя.

**Первые шаги по разработке требований к информационным системам - анализ осуществимости**

Разработка требований – это процесс, включающий мероприятия, необходимые для создания и утверждения документа, содержащего спецификацию системных требований. Для новых программных систем процесс разработки требований должен начинаться с анализа осуществимости. Началом такого анализа является общее описание системы и ее назначения, а результатом анализа — отчет, в котором должна быть четкая рекомендация, продолжать или нет процесс разработки требований проектируемой системы. Другими словами, анализ осуществимости должен осветить следующие вопросы.

1.       Отвечает ли система общим и бизнес-целям организации-заказчика и организации-разработчика?

2.         Можно ли реализовать систему, используя существующие на данный момент технологии и не выходя за пределы заданной стоимости?

3.    Можно ли объединить систему с другими системами, которые уже эксплуатируются?

Критическим является вопрос, будет ли система соответствовать целям организации. Если система не соответствует этим целям, она не представляет никакой ценности для организации. В то же время многие организации разрабатывают системы, не соответствующие их целям, либо не совсем ясно понимая эти цели, либо под влиянием политических или общественных факторов.

Выполнение анализа осуществимости включает сбор и анализ информации о будущей системе и написание соответствующего отче-та. Сначала следует определить, какая именно информация необходима, чтобы ответить на поставленные выше вопросы. Например, эту информацию можно получить, ответив на следующее:

1.       Что произойдет с организацией, если система не будет введена в эксплуатацию?

2.       Какие текущие проблемы существуют в организации и как новая система поможет их решить?

3.       Каким образом система будет способствовать целям бизнеса?

4.       Требует ли разработка системы технологии, которая до этого не использовалась в организации?

Далее  необходимо  определить  источники  информации.  Это могут быть менеджеры отделов, где система будет использоваться, разработчики программного обеспечения, знакомые с типом будущей системы, технологи, конечные пользователи и т.д.

После обработки собранной информации готовится отчет по анализу осуществимости создания системы. В нем должны быть даны рекомендации относительно продолжения разработки системы. Могут быть предложены изменения бюджета и графика работ по созданию системы или предъявлены более высокие требования к системе.

**Диаграммы модулей (диаграммы компонентов)**

В то время как другие UML диаграммы, которые описывают функциональность системы, компонентные диаграммы используются для моделирования компонентов, которые помогают сделать эти функциональные возможности.

**Диаграммы модулей** – физическая модель системы. Показывает подсистемы проектируемого приложения и основные модули каждой из них. Диаграммы модулей образуют иерархию. Для каждого модуля можно отдельно показать на диаграмме его файл-заголовок и файл реализации.

В StarUML диаграммы модулей реализуются через диаграммы компонентов.

**Диаграммы компонентов** используются для визуализации организации компонентов системы и зависимостей между ними. Они позволяют получить высокоуровневое представление о компонентах системы.

**Компонентами** могут быть программные компоненты, такие как база данных или пользовательский интерфейс; или аппаратные компоненты, такие как схема, микросхема или устройство; или бизнес-подразделение, такое как поставщик, платежная ведомость или доставка.

В StarUML диаграмма компонентов может содержать следующие элементы:

• Пакет

• Интерфейс

• Компонент – представляет собой модульную, сборную и заменимую часть системы, которая исполняется и предлагает набор интерфейсов.

• Инстанция компонента – экземпляр компонента, который постоянно находится на инстанции узла. Инстанция компонента может иметь состояние.

• Артефакт

• Порт

• Часть

• Ассоциация

• Зависимость

• Реализация

• Связь

• Соединитель

Компонентные диаграммы:

* Используются в компонентно-ориентированных разработках для описания систем с сервис-ориентированной архитектурой
* Показать структуру самого кода
* Может использоваться для фокусировки на отношениях между компонентами, скрывая при этом детализацию спецификации
* Помощь в информировании и разъяснении функций создаваемой системы заинтересованным сторонам

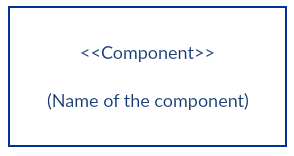
### Символы на диаграмме компонентов

Общие обозначения компонентной диаграммы, которые используются для построения компонентной диаграммы.

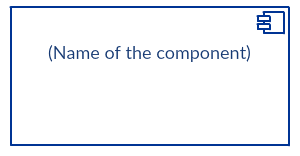
#### Компонент

Существует три способа использования символа компонента.

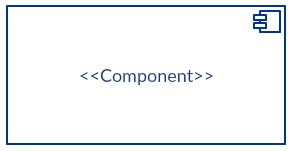
1) Прямоугольник со стереотипом компонента (текст <<компонент>>). Стереотип компонента обычно используется над именем компонента, чтобы не перепутать форму со значком класса.



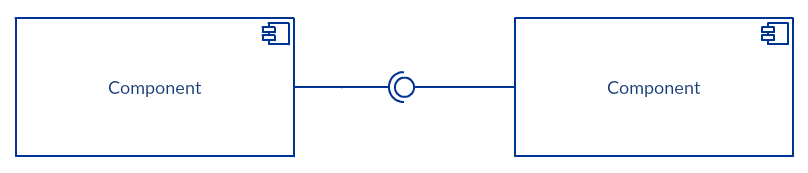
2) Прямоугольник со значком компонента в правом верхнем углу и названием компонента.



3) Прямоугольник со значком компонента и стереотипом компонента.

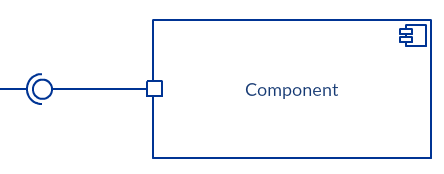


#### Предоставляемый интерфейс и требуемый интерфейс



Интерфейсы на компонентных схемах показывают, как компоненты соединены друг с другом и взаимодействуют друг с другом. Соединитель сборки позволяет соединить требуемый интерфейс компонента (представленный полукругом и сплошной линией) с предусмотренным интерфейсом (представленный окружностью и сплошной линией) другого компонента. Это показывает, что один компонент предоставляет услугу, которая требуется другому.

#### Порт



Порт (представленный маленьким квадратом в конце требуемого интерфейса или предоставляемого интерфейса) используется, когда компонент делегирует интерфейсы внутреннему классу.

#### Зависимости



Хотя вы можете показать более подробную информацию об отношениях между двумя компонентами с помощью нотации ball-and-socket (предусмотренный интерфейс и требуемый интерфейс), вы можете также использовать стрелку зависимостей, чтобы показать отношения между двумя компонентами.

### Как рисовать диаграмму компонентов

Вы можете использовать компонентную диаграмму, когда хотите представить свою систему как компоненты и показать их взаимосвязь через интерфейсы. Это помогает получить представление о внедрении системы. Ниже приведены действия, которые можно выполнить при построении компонентной диаграммы.

**Шаг 1: В**ыясните назначение диаграммы и идентифицируйте артефакты, такие как файлы, документы и т.д. в вашей системе или приложении, которые необходимо представить на диаграмме.

**Шаг 2:**По мере выяснения взаимосвязей между элементами, которые вы определили ранее, создайте ментальный макет своей компонентной диаграммы

**Шаг 3:**По мере того, как вы рисуете диаграмму, сначала добавьте компоненты, группируя их внутри других компонентов, как вам кажется подходящим

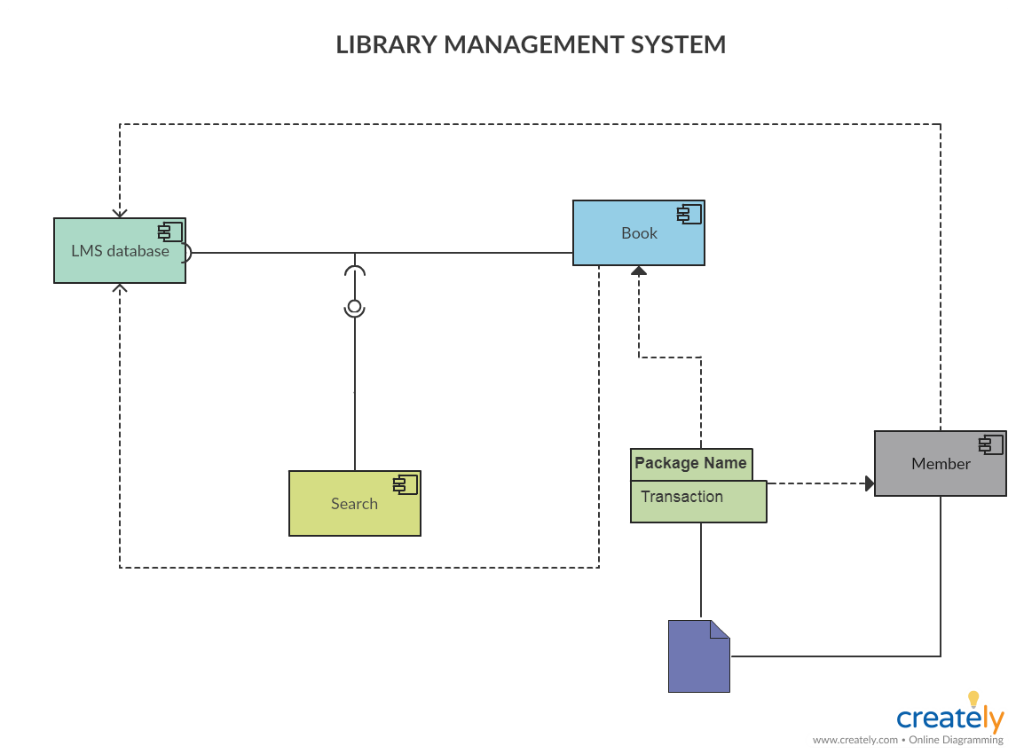
**Шаг 4:**Следующим шагом является добавление других элементов, таких как интерфейсы, классы, объекты, зависимости и т.д. в вашу компонентную диаграмму и ее завершение.

**Шаг 5:** Вы можете приложить примечания к различным частям вашей компонентной диаграммы, чтобы прояснить некоторые детали другим.

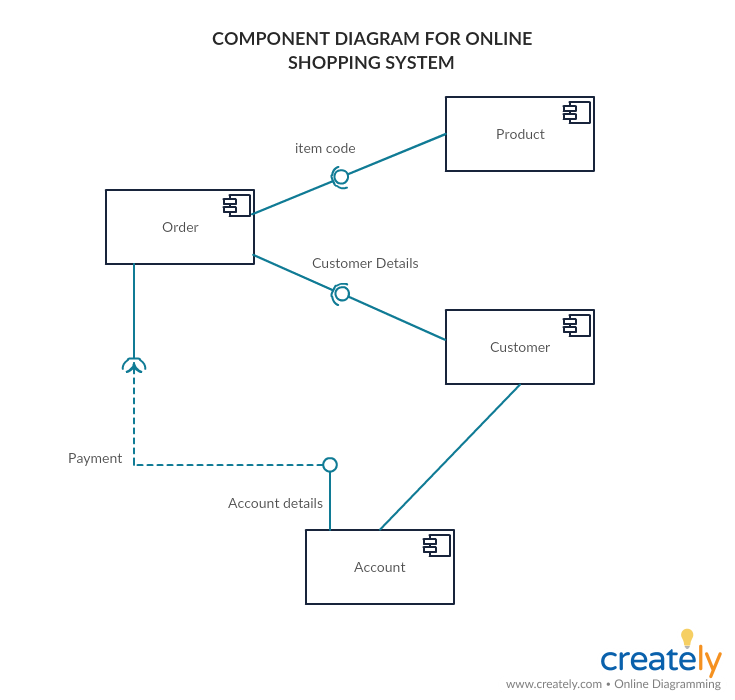
### Примеры схем компонентов

Ниже приведены шаблоны компонентных диаграмм для распространенных сценариев.

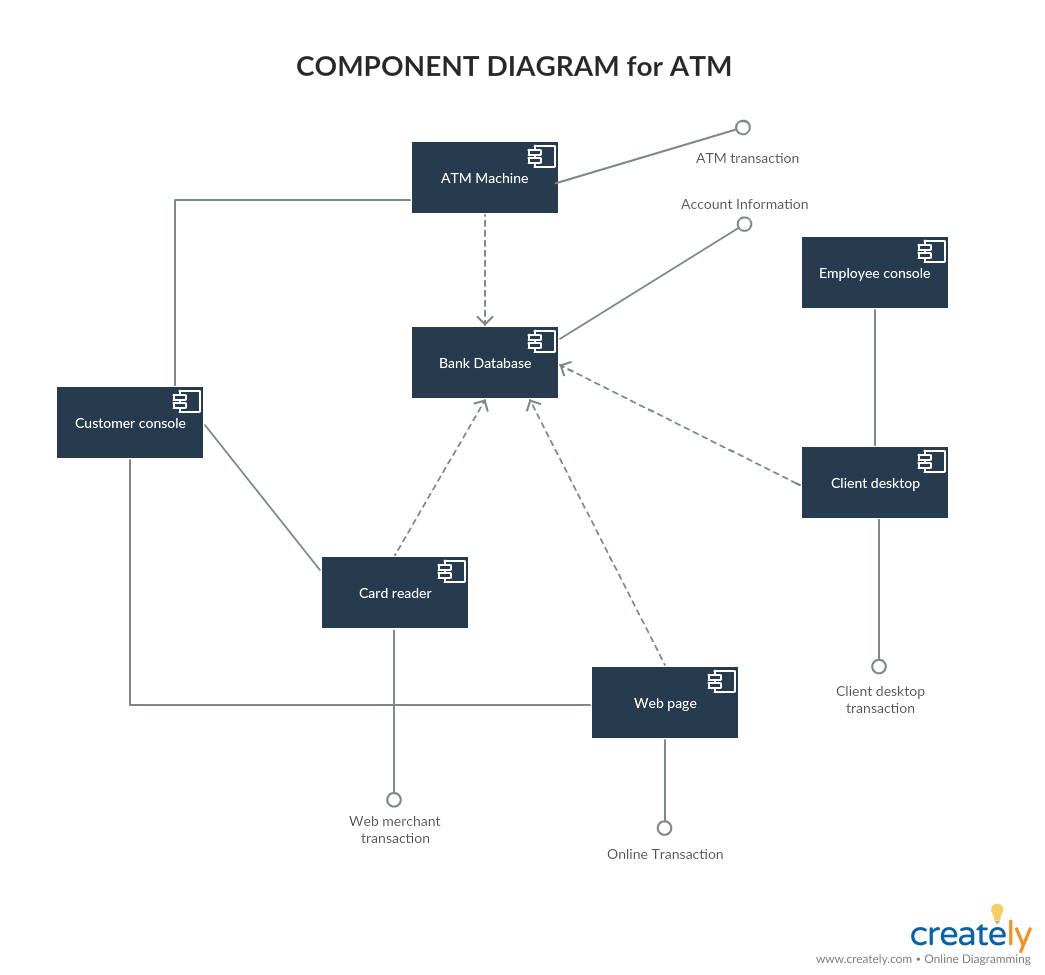
*Схема компонентов для системы управления библиотекой*

[](https://creately.com/demo-start/?tempId=jma3qob48)

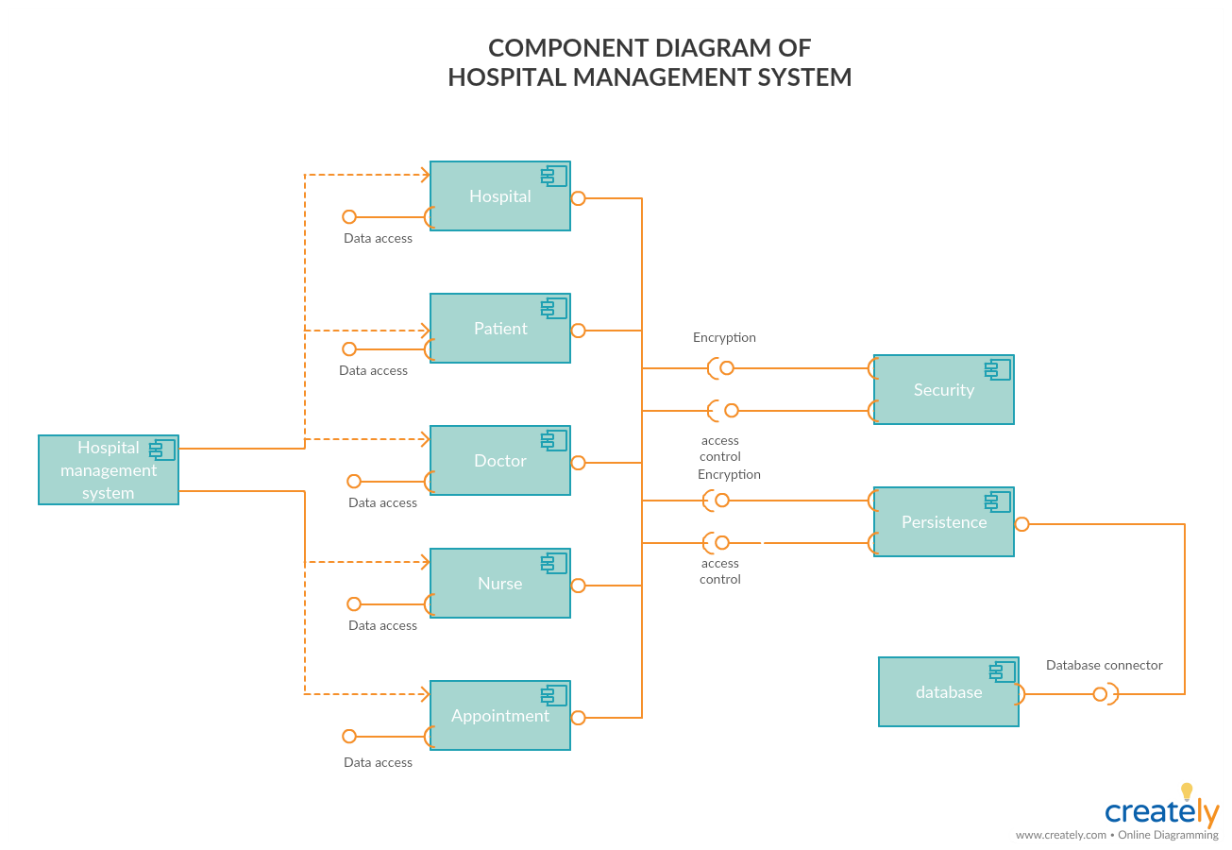
*Схема компонентов для системы онлайн-покупок*

[](https://creately.com/demo-start/?tempId=gqz0nrua1)

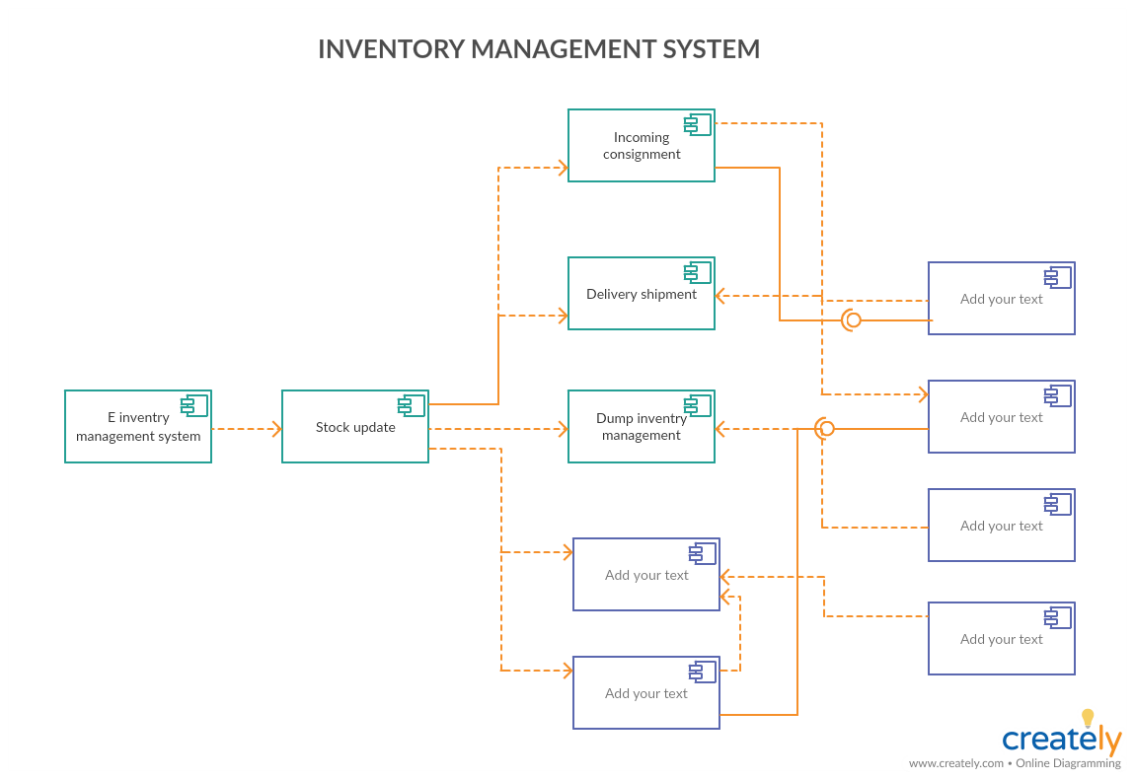
*Диаграмма компонентов для банкомата*

[](https://creately.com/demo-start/?tempId=jma3qob47)

*Схема компонентов для системы управления больницей*

[](https://creately.com/demo-start/?tempId=jma3qob46)

*Диаграмма компонентов для системы управления запасами*

[](https://creately.com/demo-start/?tempId=jma3qob49)

**2. Порядок выполнения работы**

**Задание 1.**

1) Изучить предлагаемый теоретический материал.

2) Составить подробное описание информационной системы по индивидуальному варианту.

Варианты:

1.  ИС Ресторан

2.  ИС Отдел кадров

3.  ИС Гостиница

4.  ИС Больница

5.  ИС Аптека

6.  ИС Аэропорт

7.  ИС Видеопрокат

8.  ИС Компьютерная фирма

9.  ИС Библиотека

10. ИС Туристическое агентство

11. ИС Сервисный центр

12. ИС Деканат

13. ИС Прокат автомобилей

14. ИС Ломбард

15. ИС Риэлтерская фирма

16. ИС Рекламное агентство

17. ИС Автосалон

18. ИС Продуктовый магазин

19. ИС Учет готовой продукции

20. ИС Банк

21. ИС Книжный магазин

22. ИС Магазин бытовой техники

23. ИС Успеваемость студентов

24. ИС Паспортный стол

25. ИС Склад товаров

26. ИС Кинотеатр

27. ИС Учет услуг спортивного клуба

28. ИС Косметический салон

3) На основании описания системы провести анализ осуществимости. В ходе анализа ответить на вопросы:

*Что произойдет с организацией, если система не будет введена в эксплуатацию?*

*Какие текущие проблемы существуют в организации и как новая система поможет их решить?*

*Каким образом система будет способствовать целям бизнеса?*

*Требует ли разработка системы  технологии,  которая  до этого не использовалась в организации?*

Результатом анализа должно явиться заключение о возможности реализации проекта.

**Задание 2.**

Распределить роли в группе (руководитель проекта-разработчик, системный аналитик-разработчик, тестер-разработчик).

**Задание 3.**

Заполнить разделы плана:

*Введение*

*Организация выполнения проекта*

*Анализ рисков*

Разделы должны содержать рекомендации относительно разработки системы, базовые предложения по объѐму требуемого бюджета, числу разработчиков, времени и требуемому программному обеспечению.

**Задание 4.** Построить диаграммы модулей для выбранной ИС (диаграммы компонентов).

**Задание 5.**

Составить отчет о проделанной работе.

**Содержание отчета**

В отчете следует указать:

1. Цель работы.

2. Введение. Краткое описание целей проекта и проектных ограничений (бюджетных, временных и т.д.), которые важны для управления проектом

3. Описание информационной системы (ПО) - наличие заключения о возможности реализации проекта, содержащего рекомендации относительно разработки системы, базовые предложения по объѐму требуемого бюджета, числу разработчиков, времени и требуемому программному обеспечению.

4. Анализ осуществимости, указать возможные проблемы и пути их решения.

5. Роли участников группы разработки ПО.

6. Программно-аппаратные средства, используемые при выполнении работы.

7. Заключение (выводы)

8. Список используемой литературы

**Вопросы для самоконтроля:**

1. Структура проекта.

2. Модуль и подсистемы.

3. Модульная структура проекта.

**Список литературы и ссылки на Интернет-ресурсы, содержащие информацию по теме:**

1. Вичугова, А. А. Инструментальные средства разработки компьютерных систем и комплексов : учебное пособие для СПО / А. А. Вичугова. — Саратов : Профобразование, 2020. — 135 c. — ISBN 978-5-4488-0015-3. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/66387.html (дата обращения: 04.08.2024). — Режим доступа: для авторизир. пользователей. - DOI: https://doi.org/10.23682/66387.

2. Тимофеев, А. В. Проектирование и разработка информационных систем : учебное пособие для СПО / А. В. Тимофеев, З. Ф. Камальдинова, Н. С. Агафонова. — Саратов : Профобразование, 2022. — 91 c. — ISBN 978-5-4488-1416-7. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/116285.html (дата обращения: 04.08.2024). — Режим доступа: для авторизир. пользователей. - DOI: https://doi.org/10.23682/116285.

**Методические указания к практическому занятию № 2**

**Тема: Разработка модульной структуры проекта в Visual Studio**.

**Количество часов**: 2

**Цель**:

- *обучающая*: рассмотреть особенности разработки модульной структуры проекта в Visual Studio;

- *воспитательная*: выработать умение мыслить, научить логически мыслить; оценить степень работоспособности; развивать познавательные возможности, внимание; содействовать развитию профессиональных качеств;

- *развивающая*: развивать умения и навыки применять: теорию при решении задач, навыки самостоятельной работы с методическими указаниями к практическому занятию, осуществлять самоконтроль, язык терминов.

**Задания:**

1. Изучить материал, представленный в теоретической части (1 балл).

2. Разработать модульную структуру проекта для своего варианта в Visual Studio. (4 балла).

3. Защитить выполненное задание (1 балл).

*Выводы:* выполнение практического занятия способствует формированию практических навыков по разработке модульной структуры проекта в Visual Studio.

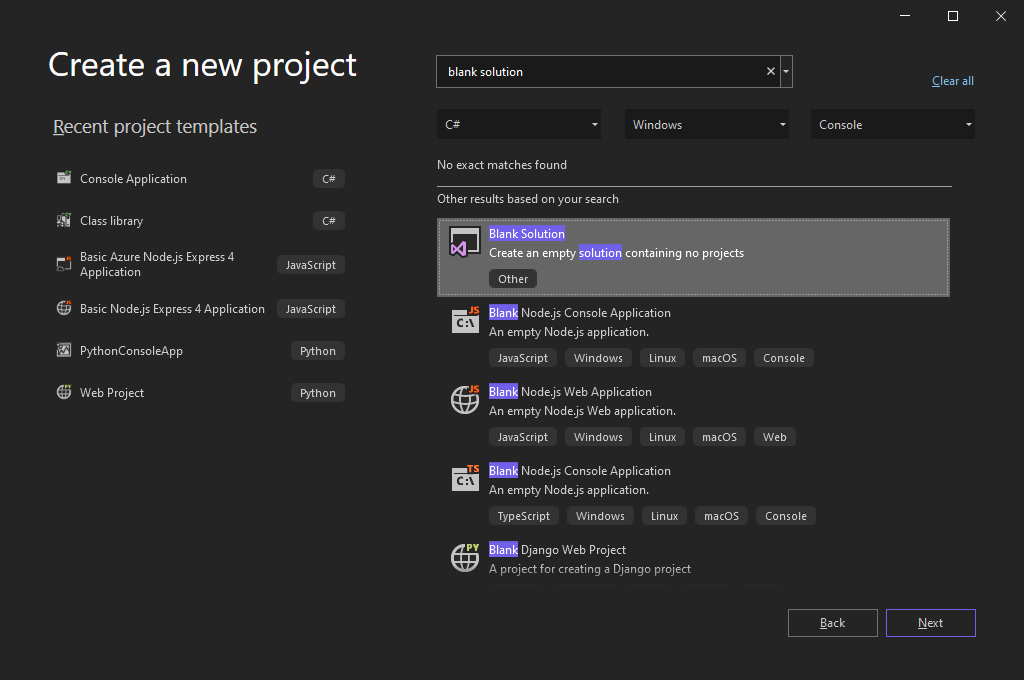
**1. Теоретическая часть**

В Visual Studio **решения** – это контейнеры, используемые для упорядочения одного проекта или нескольких связанных. Когда открывается решение в среде Visual Studio, все содержащиеся в нем проекты загружаются автоматически.

### I. Создание решения.

Обучение мы начнем с создания пустого решения. При создании проекта в среде Visual Studio она автоматически создает решение для размещения проекта, если никакое решение еще не открыто.

1. Откройте Visual Studio и в начальном окне выберите **Создать проект**.
2. На странице **Создание нового проекта** введите в поле поиска пустое решение, выберите шаблон **Пустое решение** и нажмите **Далее**.



Если на компьютере установлено несколько рабочих надстроек, шаблон **Пустое решение** может не отображаться в верхней части списка результатов поиска. Попробуйте прокрутить раздел **Другие результаты для вашего поискового запроса**, чтобы найти шаблон.

1. На странице **Настройка нового проекта** введите имя решения **QuickSolution** и выберите **Создать**.

Решение **QuickSolution** отобразится в **Обозревателе решений** в правой части окна Visual Studio.

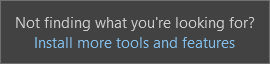
### II. Добавление проекта/

Добавим первый проект в это решение. Начните с пустого проекта и добавьте необходимые элементы.

1. Щелкните правой кнопкой мыши элемент **Решение QuickSolution** в **Обозревателе решений** и выберите **Добавить**>**Новый проект** в контекстном меню.
2. На странице **Добавление нового проекта** в поле поиска сверху введите пустой и выберите **C#** в разделе **Все языки**.
3. Выберите шаблон **Пустой проект (.NET Framework)** для C# и нажмите **Далее**.

**Примечание.** Visual Studio использует установку на основе рабочих надстроек, чтобы устанавливать только те компоненты, которые необходимы для этого типа разработки. Если вы не видите шаблон **Пустой проект (.NET Framework)**, установите рабочую нагрузку Visual Studio **Разработка классических приложений .NET**.

Простой способ установить новую рабочую надстройку при создании нового проекта – выбрать ссылку **Установка других средств и компонентов** под текстом **Не нашли то, что искали?**. В окне Visual Studio Installer выберите рабочую надстройку **Разработка классических приложений .NET** и щелкните **Изменить**.



1. На странице **Настроить новый проект** укажите имя проекта **QuickDate** и выберите **Создать**.

Проект с именем **QuickDate** появляется под решением в **Обозревателе решений**. Проект содержит **узел «Ссылки»** и один файл с именем **App.config**.

## **III. Добавление элемента в проект.**

Добавьте файл кода в пустой проект.

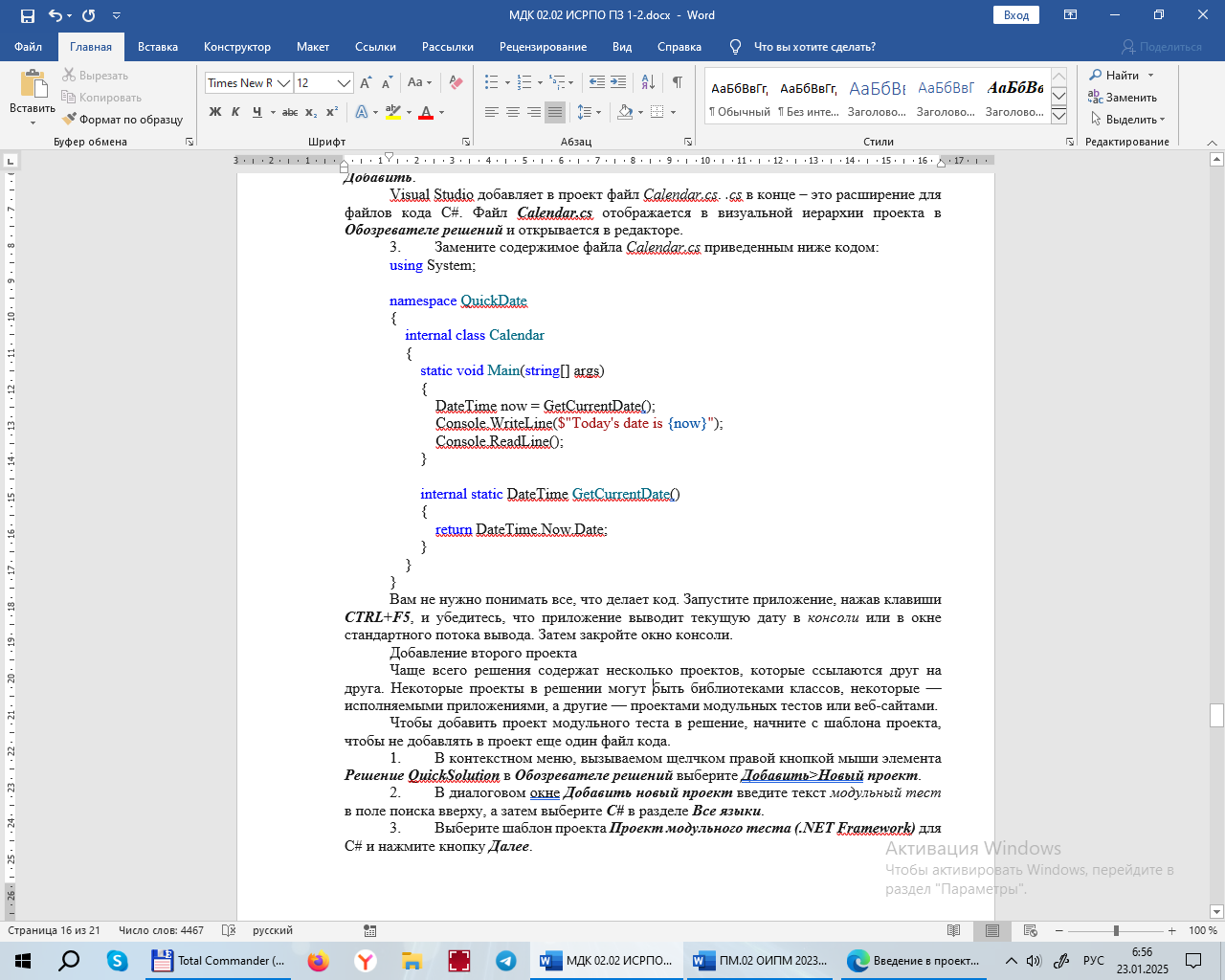
1. В контекстном меню, вызываемом щелчком правой кнопкой мыши проекта **QuickDate** в **Обозревателе решений**, выберите **Добавить**>**Новый элемент**.

Откроется диалоговое окно **Добавление нового элемента.** Выберите «**Показать все шаблоны**», если откроется диалоговое окно в компактном представлении.

1. Разверните узел **Элементы Visual C#** и выберите **Код**. В средней области выберите шаблон элемента **Класс**. В поле **Имя** введите Calendar и нажмите кнопку **Добавить**.

Visual Studio добавляет в проект файл Calendar.cs. .cs в конце – это расширение для файлов кода C#. Файл **Calendar.cs** отображается в визуальной иерархии проекта в **Обозревателе решений** и открывается в редакторе.

1. Замените содержимое файла Calendar.cs приведенным ниже кодом:



Разберите код. Запустите приложение, нажав клавиши **CTRL**+**F5**, и убедитесь, что приложение выводит текущую дату в консоли или в окне стандартного потока вывода. Затем закройте окно консоли.

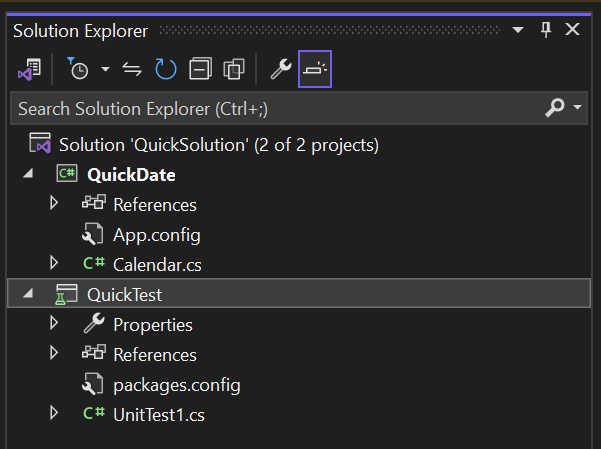
## **IV. Добавление второго проекта.**

Чаще всего решения содержат несколько проектов, которые ссылаются друг на друга. Некоторые проекты в решении могут быть библиотеками классов, некоторые –исполняемыми приложениями, а другие – проектами модульных тестов или веб-сайтами.

Чтобы добавить проект модульного теста в решение, начните с шаблона проекта, чтобы не добавлять в проект еще один файл кода.

1. В контекстном меню, вызываемом щелчком правой кнопкой мыши элемента **Решение QuickSolution** в **Обозревателе решений** выберите **Добавить**>**Новый проект**.
2. В диалоговом окне **Добавить новый проект** введите текст модульный тест в поле поиска вверху, а затем выберите **C#** в разделе **Все языки**.
3. Выберите шаблон проекта **Проект модульного теста (.NET Framework)** для C# и нажмите кнопку **Далее**.
4. На странице **Настроить новый проект** укажите имя проекта QuickTest и выберите **Создать**.

Visual Studio добавляет проект **QuickTest** в **Обозреватель решений**, а файл **UnitTest1.cs** открывается в редакторе.

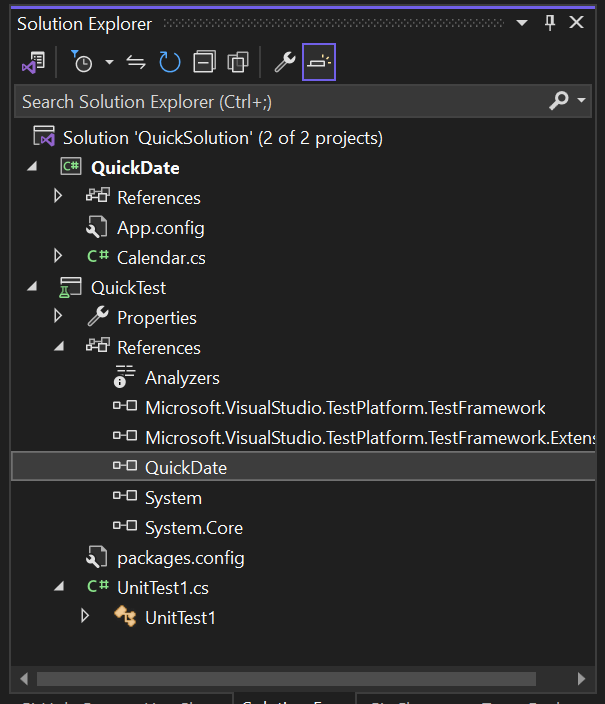


## **V. Добавление ссылки на проект.**

Мы будем использовать новый проект модульного теста для тестирования своего метода в проекте **QuickDate**, поэтому нужно добавить ссылку на **QuickDate** в проект **QuickTest**. Эта ссылка создает зависимость сборки между двумя проектами, то есть **QuickDate** будет собран перед **QuickTest** при сборке решения.

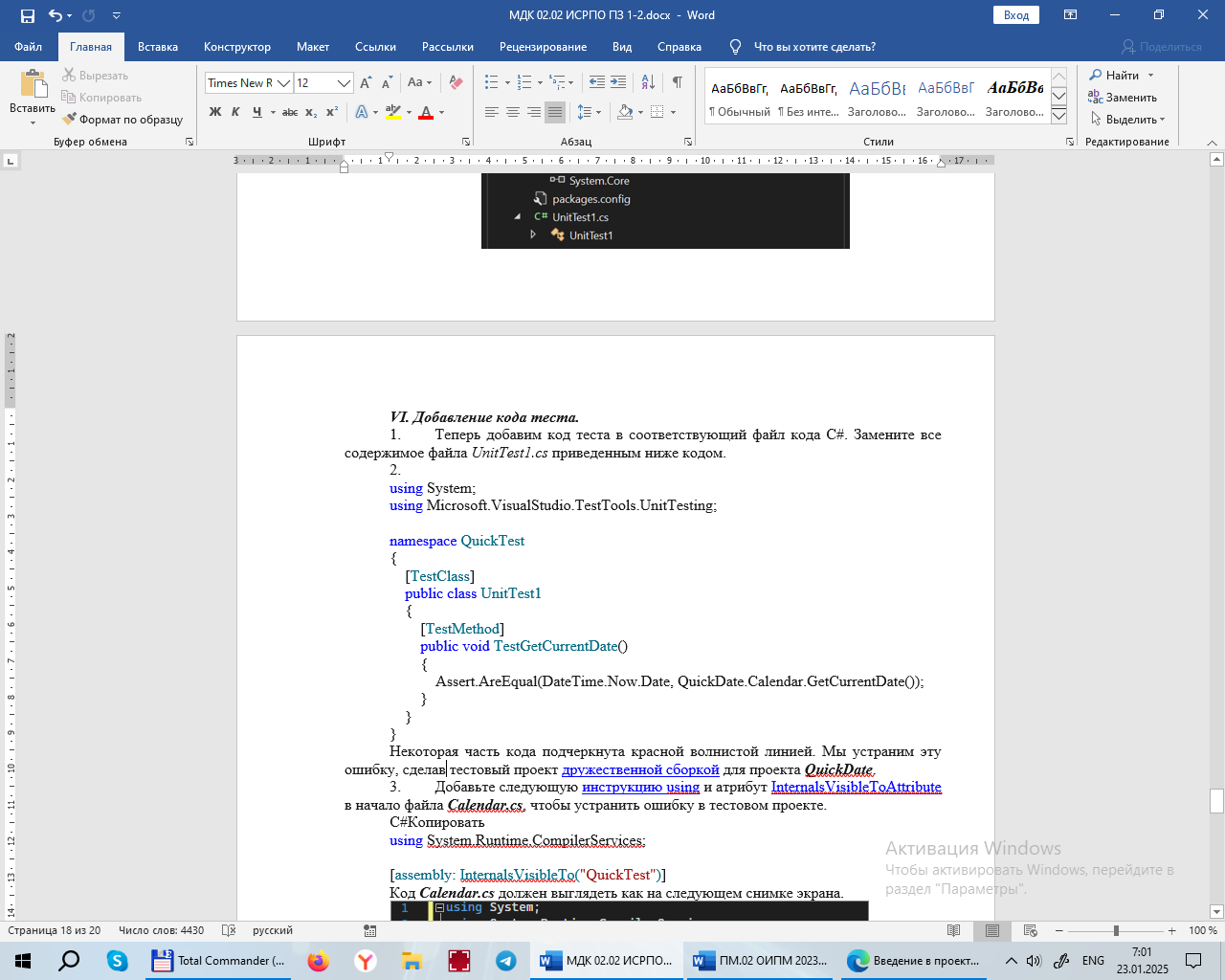
1. В **обозревателе решений** щелкните узел **Ссылки** в проекте **QuickTest** правой кнопкой мыши и выберите в контекстном меню пункт **Добавить ссылку**.
2. **В диалоговом окне «Диспетчер ссылок»** выберите **«Проекты»**. В средней области установите флажок рядом с пунктом **QuickDate** и нажмите **ОК**.

Ссылка на проект **QuickDate** отображается в проекте **QuickTest** в **Обозревателе решений**.



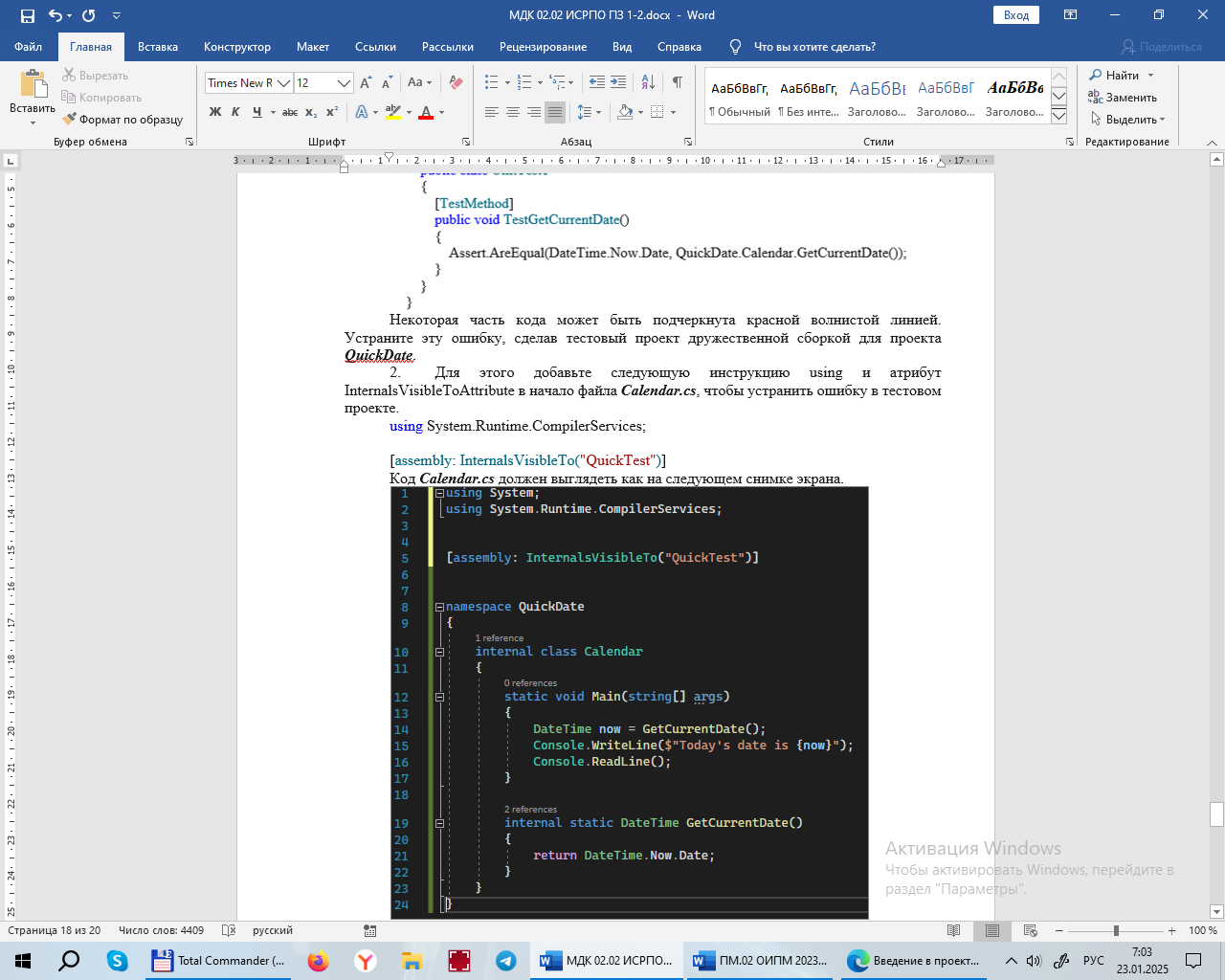
## **VI. Добавление кода теста.**

1. Теперь добавим код теста в соответствующий файл кода C#. Замените все содержимое файла UnitTest1.cs приведенным ниже кодом.

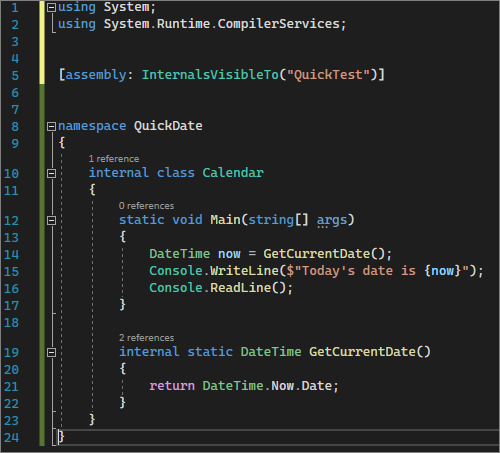


Некоторая часть кода может быть подчеркнута красной волнистой линией. Устраните эту ошибку, сделав тестовый проект дружественной сборкой для проекта **QuickDate**.

1. Для этого добавьте следующую инструкцию using и атрибут InternalsVisibleToAttribute в начало файла ***Calendar.cs***, чтобы устранить ошибку в тестовом проекте.

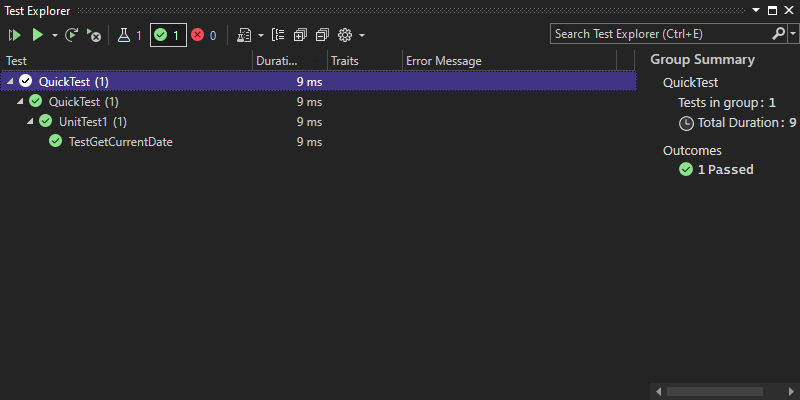


Код **Calendar.cs** должен выглядеть как на следующем снимке экрана.



### VII. Выполнить модульный тест.

Чтобы проверить работоспособность модульного теста, выберите **Тест**>**Запустить все тесты** в строке меню. Открывается окно **Обозреватель тестов**, где должно быть указано, что тест **TestGetCurrentDate** пройден.



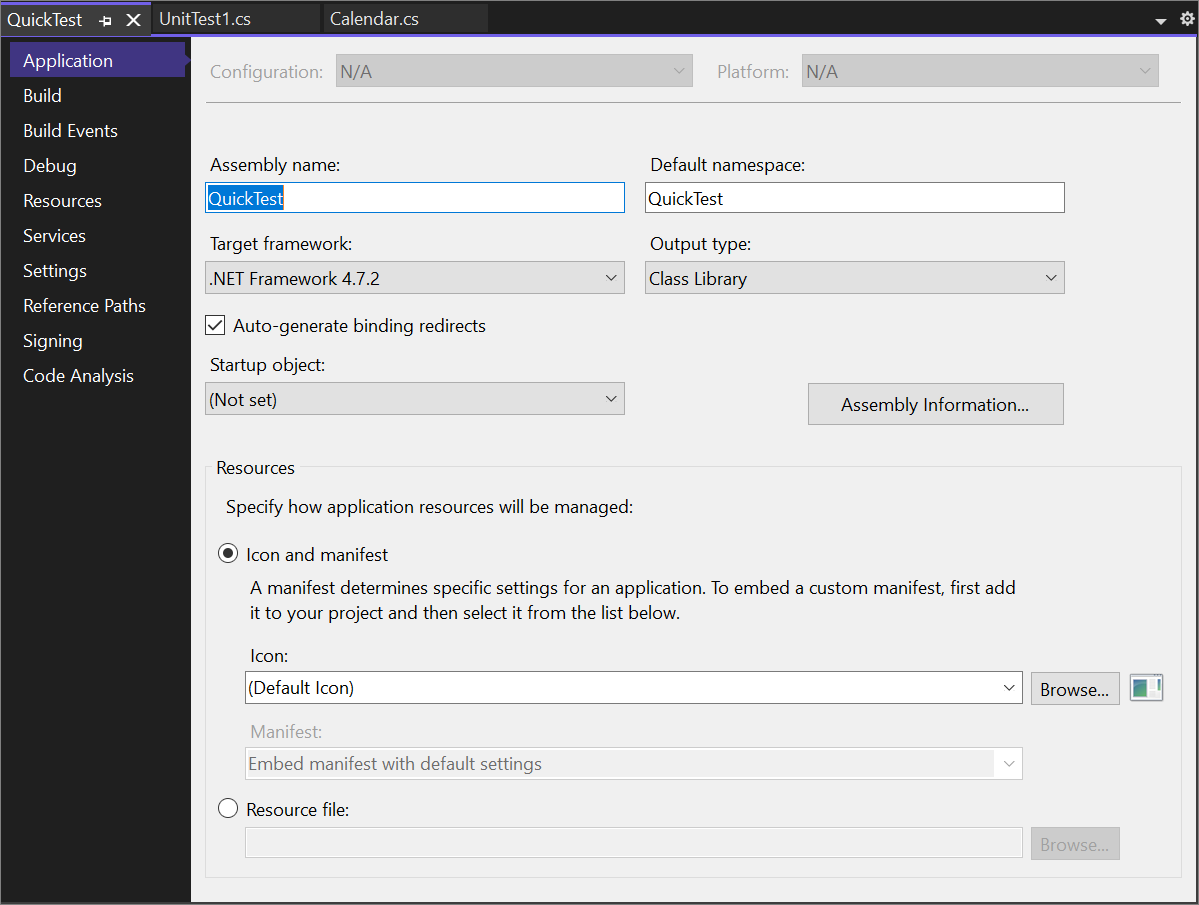
## **VIII. Свойства проекта.**

Строка в файле Calendar.cs, содержащая атрибут InternalsVisibleToAttribute, ссылается на имя сборки (имя файла) проекта **QuickTest**. Имя сборки может не всегда совпадать с именем проекта. Чтобы найти имя сборки проекта, откройте свойства проекта. Страницы свойств содержат различные параметры для проекта.

1. В **Обозревателе решений** щелкните правой кнопкой мыши проект **QuickTest** и выберите **Свойства** или выберите проект и нажмите клавиши **ALT**+**Enter**.

Страницы свойств проекта открываются на вкладке **Приложение**. **Имя сборки** проекта QuickTest – действительно **QuickTest**.

При необходимости это имя можно изменить здесь. При сборке тестового проекта имя итогового двоичного файла изменится с QuickTest.dll на <NewName>.dll.



1. Изучите другие вкладки на страницах свойств проекта, такие как **Сборка** и **Отладка**. Эти вкладки отличаются для различных типов проектов.