Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

**АНАЛИЗ ПРОГРАММНОГО КОДА НА СООТВЕТСТВИЕ ПРИНЦИПАМ GRASP**

**LOW COUPLING И HIGH COHESION**

Отчет по лабораторной работе номер 2

по дисциплине «Технология разработки программного обеспечения»

Студент гр. 588-М1

\_\_\_\_\_\_\_\_\_\_ А. А. Усольцева

\_\_\_\_\_\_\_\_\_\_

Руководитель

К.т.н., доцент каф. КСУП

\_\_\_\_\_\_\_\_\_\_ А. А. Калентьев

\_\_\_\_\_\_\_\_\_\_

**Оглавление**

[1 Введение 3](#_Toc6085250)

[2 Анализ программного кода 4](#_Toc6085251)

[2.1 Низкая связность (Low Coupling) 4](#_Toc6085252)

[2.1.1 Связность по содержимому 4](#_Toc6085253)

[2.1.2 Связность по глобальным данным 4](#_Toc6085254)

[2.1.3 Связность по внешним ссылкам 4](#_Toc6085255)

[2.1.4 Связность по управлению 5](#_Toc6085256)

[2.1.5 Связность по образцу 5](#_Toc6085257)

[2.1.6 Связность по параметрам 5](#_Toc6085258)

[2.1.7 Связность по сообщениям 5](#_Toc6085259)

[2.1.8 Связность отсутствует 6](#_Toc6085260)

[2.2 Высокое зацепление (High Cohesion) 6](#_Toc6085261)

[2.2.1 Случайное зацепление 6](#_Toc6085262)

[2.2.2 Логическое зацепление 6](#_Toc6085263)

[2.2.3 Временное зацепление 7](#_Toc6085264)

[2.2.4 Процедурное зацепление 7](#_Toc6085265)

[2.2.5 Коммуникационное/информационное зацепление 7](#_Toc6085266)

[2.2.6 Последовательное зацепление 7](#_Toc6085267)

[2.2.7 Функциональное зацепление 8](#_Toc6085268)

[2.2.8 Зацепление по данным 8](#_Toc6085269)

[3 Заключение 9](#_Toc6085270)

# **1 Введение**

Шаблоном проектирования или паттерном в разработке программного обеспечения называется повторяемая архитектурная конструкция, которая представляет собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

GRASP – (сокращенно от англ. General Responsibility Assignment Software Patterns – общие образцы распределения обязанностей) – это паттерны, используемые в ООП для решения общих задач по назначению обязанностей классам и объектам.

В книге Крэга Лармана «Применение UML и шаблонов проектирования» описано девять таких шаблонов: каждый помогает решить некоторую проблему, возникающую как в объектно-ориентированном анализе, так и в практически любом проекте по разработке программного обеспечения. Таким образом, шаблоны GRASP – это хорошо документированные, стандартизированные и проверенные временем принципы объектно-ориентированного анализа, а не попытка привнести что-то принципиально новое.

В данной работе будут рассмотрены следующие два паттерна проектирования:

– Слабое зацепление (Low Coupling);

– Сильная связность (High Cohesion).

# **2 Анализ программного кода**

## **2.1 Низкая связность (Low Coupling)**

Степень связности – это мера, определяющая, насколько жестко один элемент связан с другими элементами, либо каким количеством данных о других элементах он обладает.

Высокая связность связана со следующими проблемами:

– изменение в связанных классах приводят к локальным изменения в данном классе;

– затрудняется понимание каждого класса в отдельности;

– усложняется повторное использование, поскольку для этого требуется дополнительный анализ классов, с которыми связан данный класс.

Для связности есть своя шкала. Далее рассмотрим типы связности.

### **2.1.1 Связность по содержимому**

Связность по содержимому (от англ. Content coupling). Данная связность является самой высокой, то есть худшей. Она происходит, когда один из модулей изменяет локальные данные в другом модуле. Это затрудняет понимание и осознание смысла программы.

Например для смены состояний в программе у модуля EditContext из разных мест программы вызывается метод SetActiveState(States state /\*Enum\*/), что влечет изменение локальных данных модуля.

SetActiveState(States.States.SelectionState);

В классе ControlUnit для выделения вставленной фигуры в классах состояний для перехода в другие состояния есть подобные вызовы, и в GUI части программы.

### **2.1.2 Связность по глобальным данным**

Связность по глобальным данным (от англ. Common coupling). Она происходит, когда два модуля связаны через общие глобальные структуры данных.

Изменение глобальной переменной в одном объекте приведет к изменению поведения всех объектов системы, поскольку глобальные переменные принадлежат состоянию каждого объекта системы.

Решением описанной проблемы является создание специализированного класса-администратора, который инкапсулирует в себе все глобальные переменные и будет предоставлять ограниченный доступ к данным.

Например, для передачи некоторых данных из состояния в состояние используются глобальные переменные в рамках модуля состояний EditContext. Так, программа, после добавления фигуры из состояния «Добавить фигуру», ссылается на только что добавленную фигуру. Производится переход в следующее состояние «Добавление точки в фигуру». Данное состояние должно знать, в какую фигуру следует добавлять точки, так оно использует EditContext.ActiveFigure, что делает его глобальной переменной в модуле состояний.

### **2.1.3 Связность по внешним ссылкам**

Связность по внешним ссылкам (от англ. External coupling). Данная связность происходит, когда два модуля разделяют:

– общий формат данных, описанный в ином месте;

– протокол обмена данными;

– интерфейс устройства.

Например, Enum States (перечисление состояний) – общий формат данных, который используется в нескольких частях программы. Используется в EditContext, в конкретных состояниях, в GUI части программы, из которой пользователь может переключать состояния.

### **2.1.4 Связность по управлению**

Связность по управлению (от англ. Control coupling). Она происходит, когда один модуль должен выполнить некие операции в порядке, который определяется другим модулем.

Решением данной проблемы является вынесение общей последовательности действий в отдельную абстракцию.

В качестве примера можно привести добавление команды в стек команд и её дальнейшее выполнение. Данную операцию можно увидеть во многих частях программы:

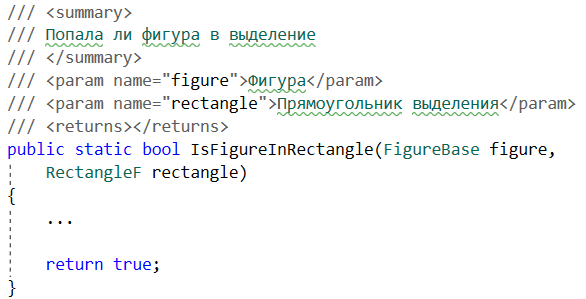
ControlUnit.StoreCommand(command);

ControlUnit.Do();

### **2.1.5 Связность по образцу**

Связность по образцу (от англ. Stamp coupling или также Data-structured coupling). Данная связность появляется, когда модули разделяют структуру данных и используют только ее части, возможно, разные части (например, передавая всю запись в функцию, которой нужна только ее часть).

Например, в метод FigureEditor::IsFigureInRectangle передается объект типа FigureBase, хотя используется только часть этой структуры, а именно PointsSettings.



На данный момент данная особенность программы исправлена: метод IsFigureInRectangle перенесен в PointsSettings.

### **2.1.6 Связность по параметрам**

Связность по параметрам (от англ. Data coupling). Данный вид связности является наиболее желательным. Это самый простой и очевидный тип связности, который обеспечивает легкость понимания и модификации.

Данная связность происходит, когда один модуль должен пользоваться услугами и процедурами другого модуля, и при этом единственная связь между ними осуществляется через входные и выходные параметры.

Например, в модуль отрисовки передается сама фигура как параметр и поверхность отрисовки.

DrawerFactory::DrawFigure (shape, graphics)

### **2.1.7 Связность по сообщениям**

Связность по сообщениям (от англ. Message coupling). В данном виде связности обмен между модулями осуществляется только путем вызова методов (передачи сообщений).

В программе, например, данный тип связи используется только в виде подписки на событие OnPaint в WinForms для отрисовки фигур.

### **2.1.8 Связность отсутствует**

Также одним из видов связности определен как отсутствие связности (от англ. No coupling). Данная связность не входит в шкалу типов связности и указывает, что рассматриваемые модули не связаны друг с другом.

## **2.2 Высокое зацепление (High Cohesion)**

Зацепление – это мера связности и сфокусированности обязанностей класса. Используется для оценки возможных альтернатив.

Низкое зацепление связано со следующими проблемами:

– трудность понимания;

– сложность при повторном использовании;

– сложности поддержки;

– ненадёжность, постоянная подверженность изменениям.

Для зацепления также есть своя шкала. Далее рассмотрим типы зацепления.

### **2.2.1 Случайное зацепление**

Случайное зацепление (от англ. Coincidental cohesion). Данный тип зацепления подразумевает, что элементы модуля группируются без видимой причины. Это нарушает принципы объектно-ориентированного программирования.

Класс FigureEditor, диаграмма которого представлена на рисунке 2.1, например, группирует в себе разные обязанности как Utility класс, хотя некоторые его обязанности очевидно должны относиться либо к PointsSettings, либо к FigureBase, поскольку данные классы обладают достаточными для этого данными.

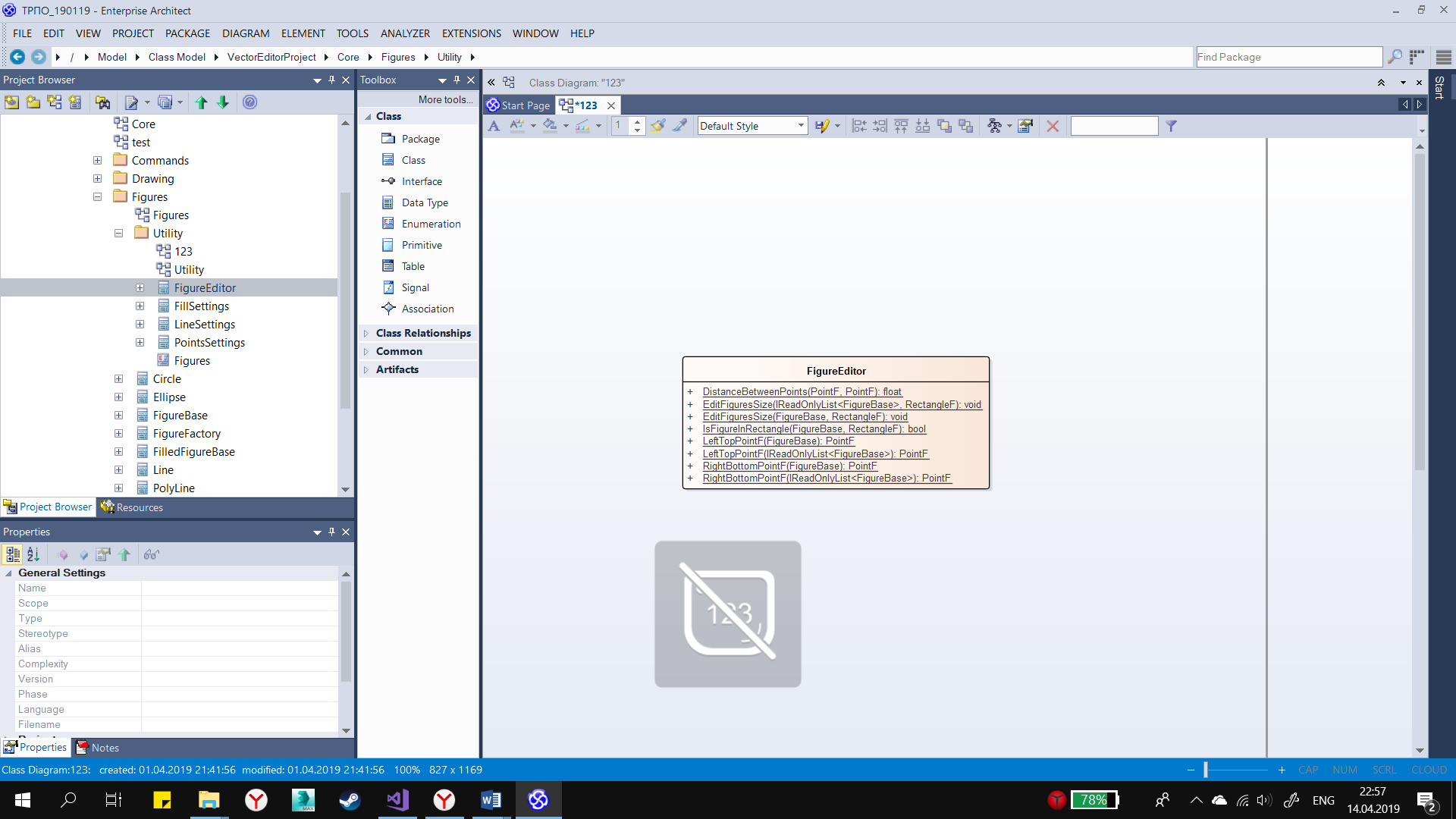


Рисунок 2.1 – Диаграмма класс FigureEditor

### **2.2.2 Логическое зацепление**

Логическое зацепление (от англ. Logical cohesion). Подразумевается, что имеется логическая группировка между элементами модуля (или методами класса), но нет фактического соединения ни по данным, ни по управлению.

Данный вид зацепления не всегда является ошибкой: например, библиотека математических функций – не содержит в себе данных, математические функции преимущественно независимы друг от друга.

Так в программе методы обработки пользовательского ввода и отрисовки состояния логически объединены в классе StateBase, диаграмма которого представлена на рисунке 2.2, но они различны по своей природе и назначению.

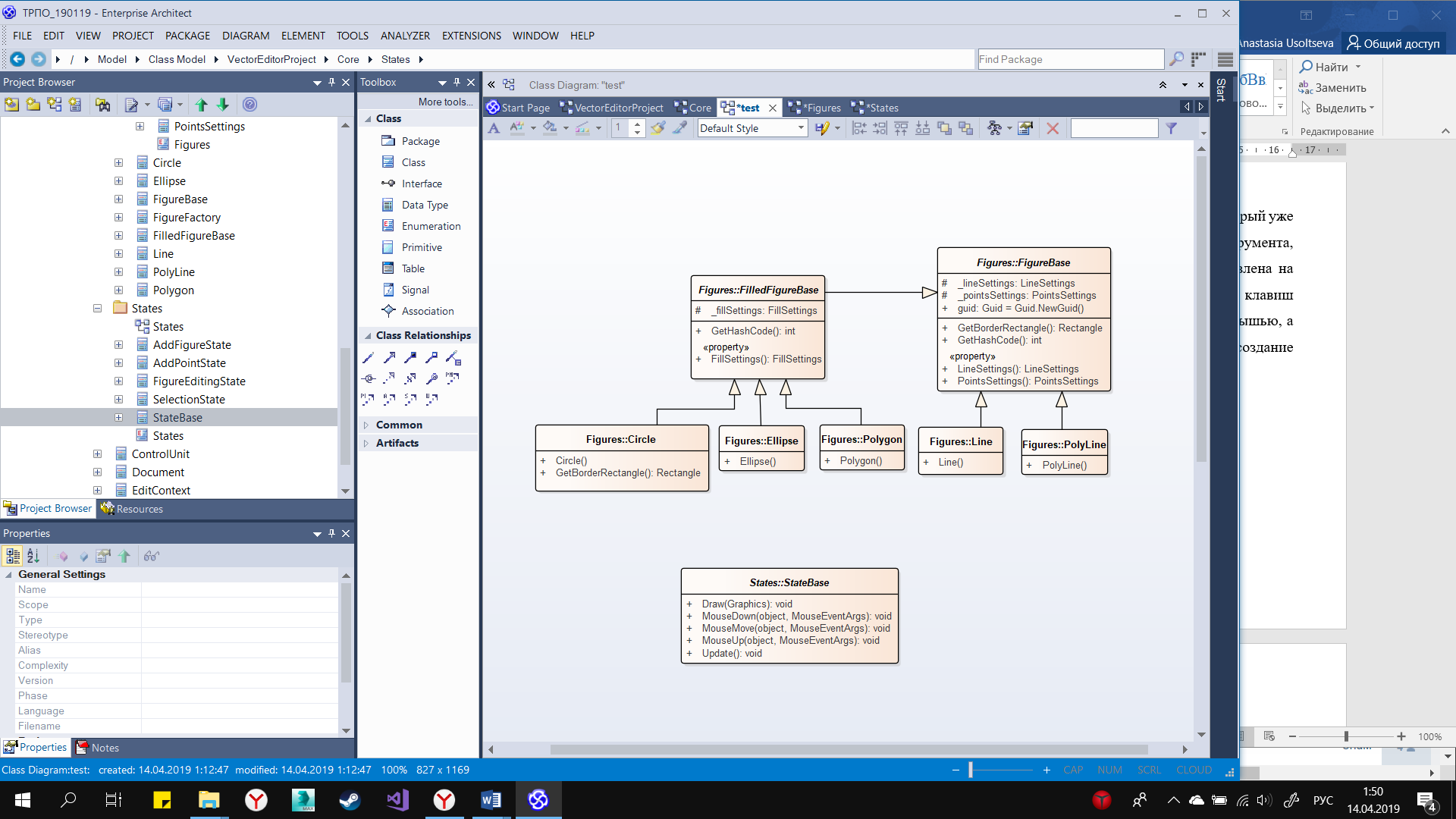


Рисунок 2.2 – Диаграмма класса StateBase

### **2.2.3 Временное зацепление**

Временное зацепление (от англ. Temporal cohesion). В данном виде зацепления элементы объединяются вместе, так как все они должны использоваться примерно одновременно.

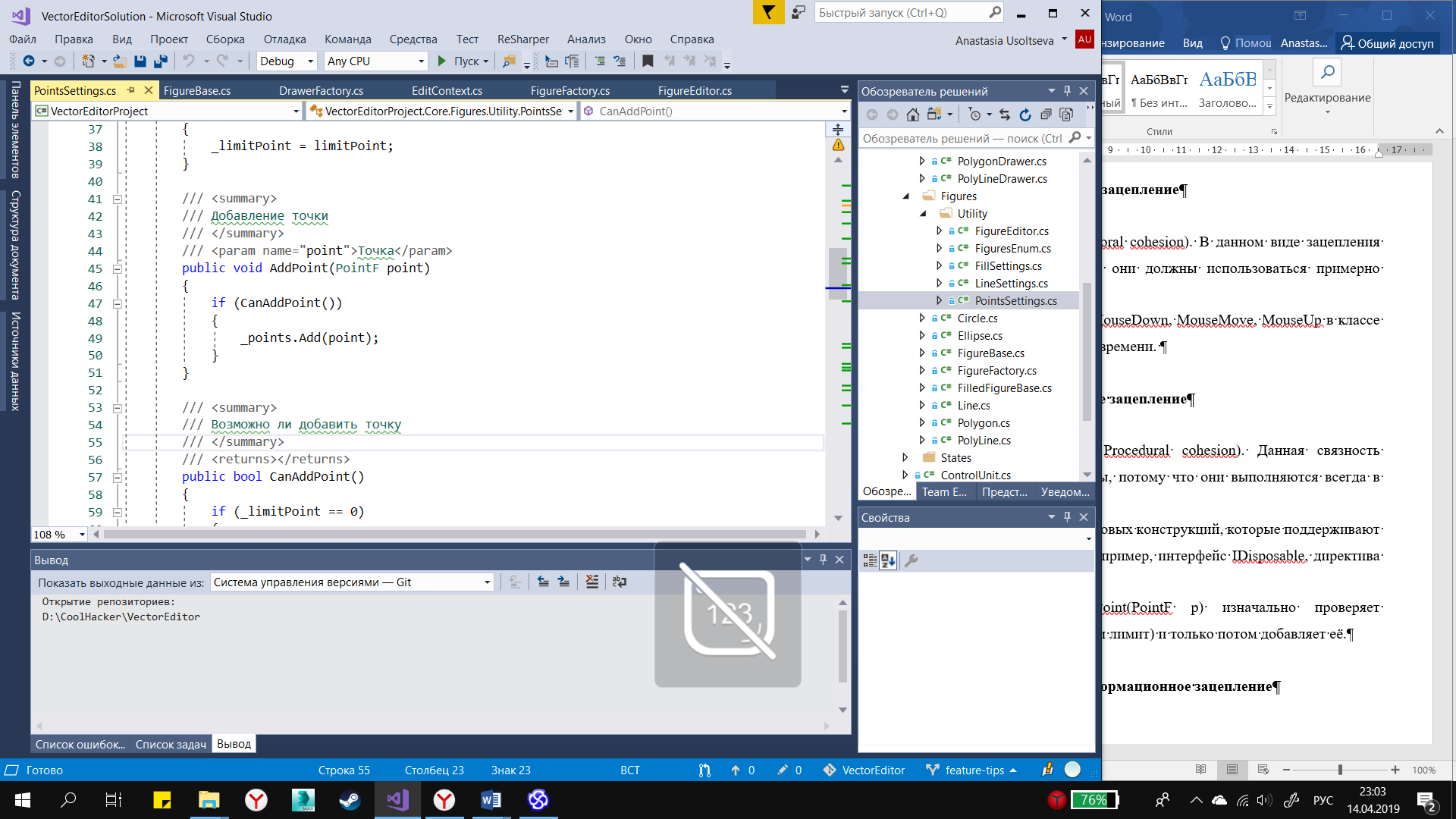
Например, методы обработки событий MouseDown, MouseMove, MouseUp в классе StateBase (рисунок 2.2) связаны логически и по времени.

### **2.2.4 Процедурное зацепление**

Процедурное зацепление (от англ. Procedural cohesion). Данная связность появляется, когда части модуля сгруппированы, потому что они выполняются всегда в определенной последовательности.

Решением является использование языковых конструкций, которые поддерживают правильный порядок выполнения процедур (например, интерфейс IDisposable, директива using).

Так метод PointsSettings::AddPoint(PointF p) изначально проверяет возможность добавления точки (не превышен ли лимит) и только потом добавляет её.



### **2.2.5 Коммуникационное/информационное зацепление**

Коммуникационное / информационное зацепление (от англ. Communicational / informational cohesion). В данном виде зацепления элементы объединены в модуль, поскольку они имеют доступ к одним и тем же устройствам ввода/вывода. Модуль работает как администратор устройства.

Примером, вероятно (хотя и абсолютно не уверенно), можно назвать DrawerFactory, диаграмма которого представлена на рисунке 2.3. Данный класс объединяет в себе отрисовщики фигур, но в нем нет ресурса ввода-вывода, если не считать рисование на Graphics. Других примеров найти не удалось.

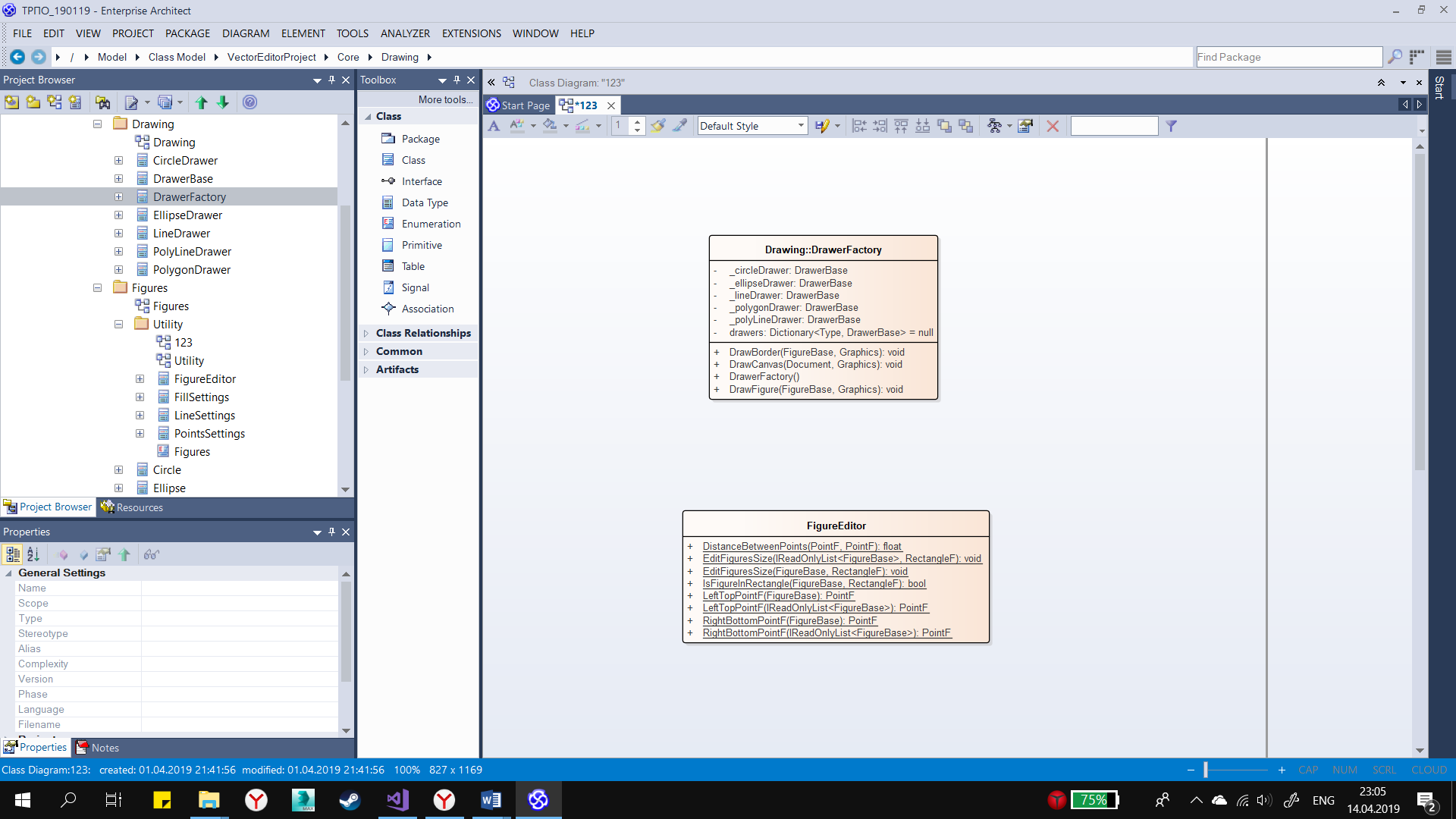


Рисунок 2.3 – Диаграмма класса DrawerFactory

### **2.2.6 Последовательное зацепление**

Последовательное зацепление (от англ. Sequential cohesion). Данный вид связности подразумевает, что элементы модуля должны активизироваться в определенном порядке – входные данные одного модуля является выходными другого (не то же, что и временное зацепление).

Пример 1: добавление команды в стек команд и её дальнейшее выполнение, данную операцию можно увидеть во многих частях программы:

ControlUnit.StoreCommand(command);

ControlUnit.Do();

Пример 2: описанный в пункте 2.1.2 настоящего отчета приведен пример, подходящий и для описания данного вида зацепления. Так, одно состояние создает фигуру и на выходе имеется фигура, она же идет на вход при переходе в состояние добавления точки.

### **2.2.7 Функциональное зацепление**

Функциональное зацепление (от англ. Functional cohesion) является желательным видом зацепления. Здесь все элементы модуля связаны выполнением единой задачи.

Класс CommandBase имеет всего 2 метода: Do() и Undo(). Которые, можно сказать, направлены на решение одной задачи – возможности наката и отката изменений.

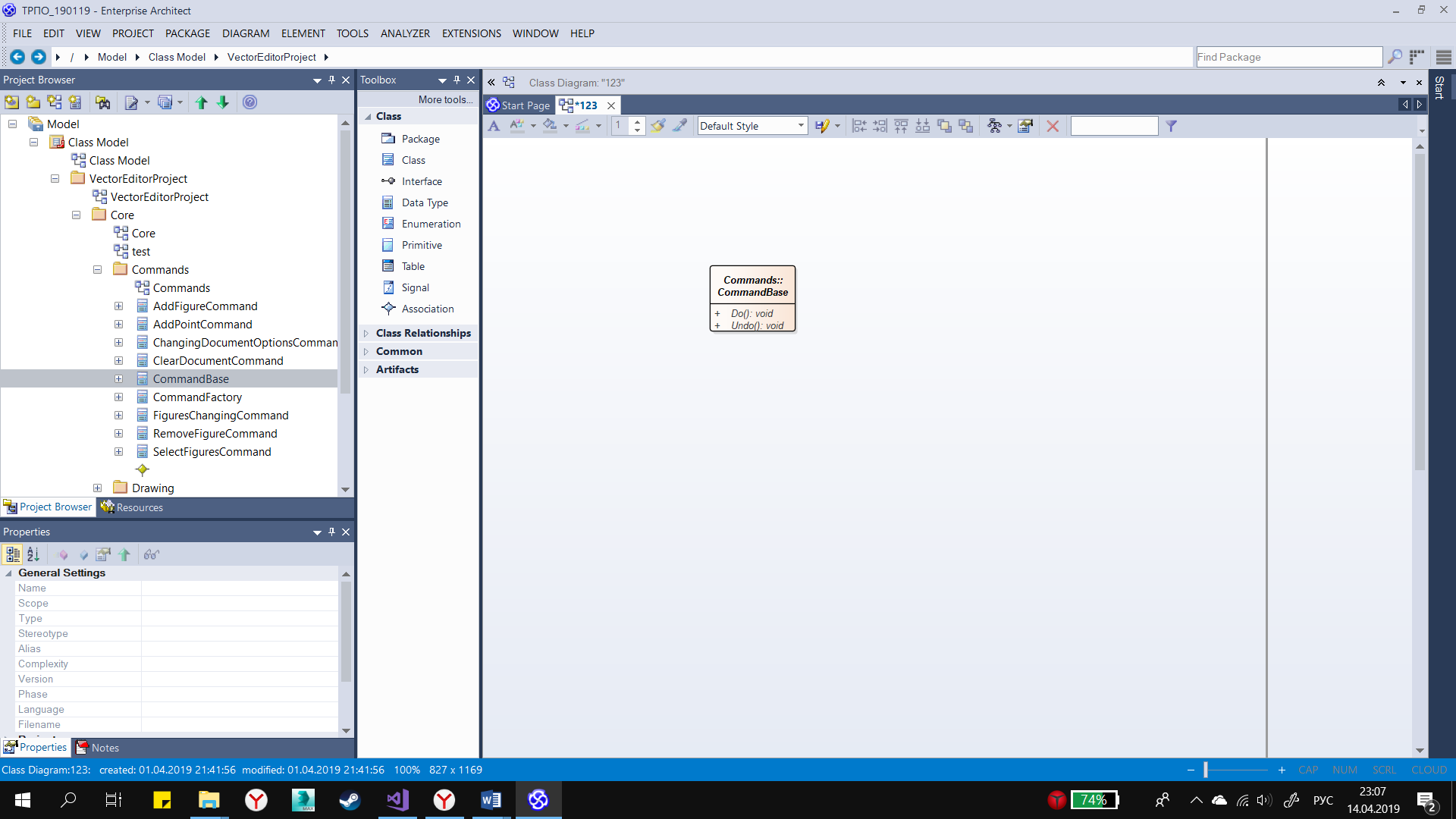


Рисунок 2.4 – Диаграмма класса CommandBase

### **2.2.8 Зацепление по данным**

Зацепление по данным (от англ. Data cohesion) также является желательным видом зацепления, где все элементы модуля также связаны выполнением единой задачи.

Связность на уровне данных возникает в модуле, когда он внутренним образом определяет набор данных и экспортирует подпрограммы (процедуры, функции, методы), которые манипулируют этой структурой данных.

Так в программе классы конкретных фигур содержат только данные для фигур, как показано на рисунке 2.5.

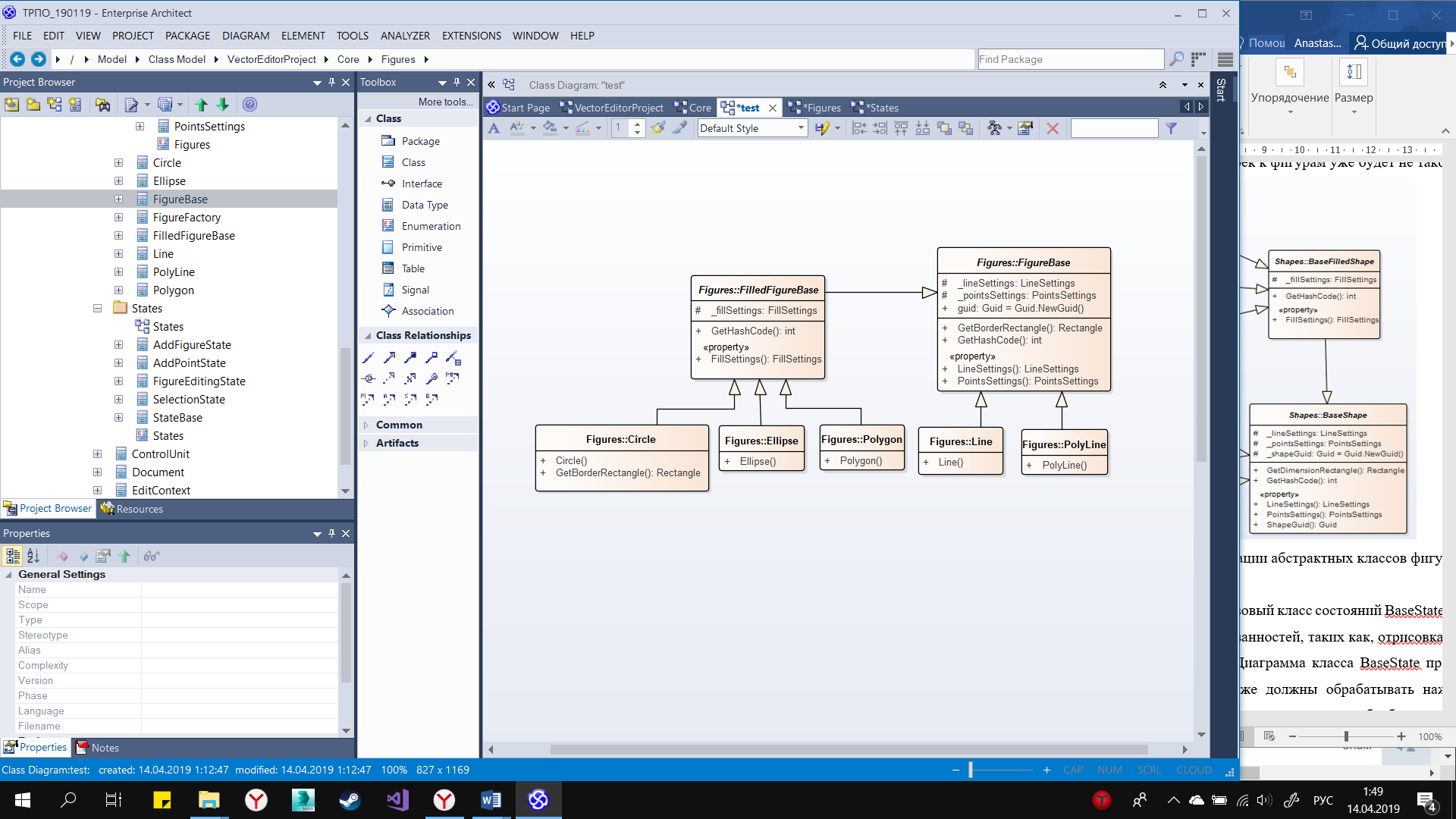


Рисунок 2.5 – Диаграмма классов фигур

# **3 Заключение**

В ходе работы был произведен анализ программного кода на соответствие паттернам GRASP, а именно таким двум шаблонам, как Низкая связность (от англ. Low Coupling) и Высокое зацепление (от англ. High Cohesion).

В работе рассмотрена шкала связности для обоих шаблонов. Приведено описание всех типов связности и зацепления, приведены примеры положительного и негативного опыта использования представленных типов связности и зацепления.