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**

Отчет по лабораторной работе номер 3

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](#_Toc6089836)

[2 Анализ программного кода 4](#_Toc6089837)

[2.1 Информационный эксперт 4](#_Toc6089838)

[2.2 Создатель 4](#_Toc6089839)

[2.3 Контроллер 5](#_Toc6089840)

[2.4 Полиморфизм 6](#_Toc6089841)

[2.5 Чистая выдумка 7](#_Toc6089842)

[2.6 Посредник 8](#_Toc6089843)

[2.7 Сокрытие реализации 8](#_Toc6089844)

[3 Заключение 9](#_Toc6089845)

# **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 предыдущей лабораторной работе были рассмотрены такие два паттерна проектирования GRASP как Слабое зацепление (Low Coupling) и Сильная связность (High Cohesion). В данной работе будут рассмотрены оставшиеся семь паттернов проектирования GRASP.

# **2 Анализ программного кода**

## **2.1 Информационный эксперт**

Информационный эксперт (от англ. Information Expert). «Ответственность должна быть назначена тому, кто владеет максимумом необходимой информации для исполнения – информационному эксперту». Данный шаблон определяет базовый принцип распределения ответственностей.

Локализация ответственностей, проводимая согласно шаблону, повышает инкапсуляцию, простоту восприятия и готовность компонентов к повторному использованию, однако расширение круга обязанностей классов приводит к снижению зацепления и повышению связности.

Класс DrawerFactory можно назвать положительным примером применения данного шаблона. У класса имеется все необходимое для того, чтобы на заданной поверхности нарисовать требуемую фигуру.

Класс FigureEditor можно привести в качестве отрицательного примера. Диаграмма данного класса представлена на рисунке 2.1. FigureEditor – Utility класс, который вообще не содержит данных.

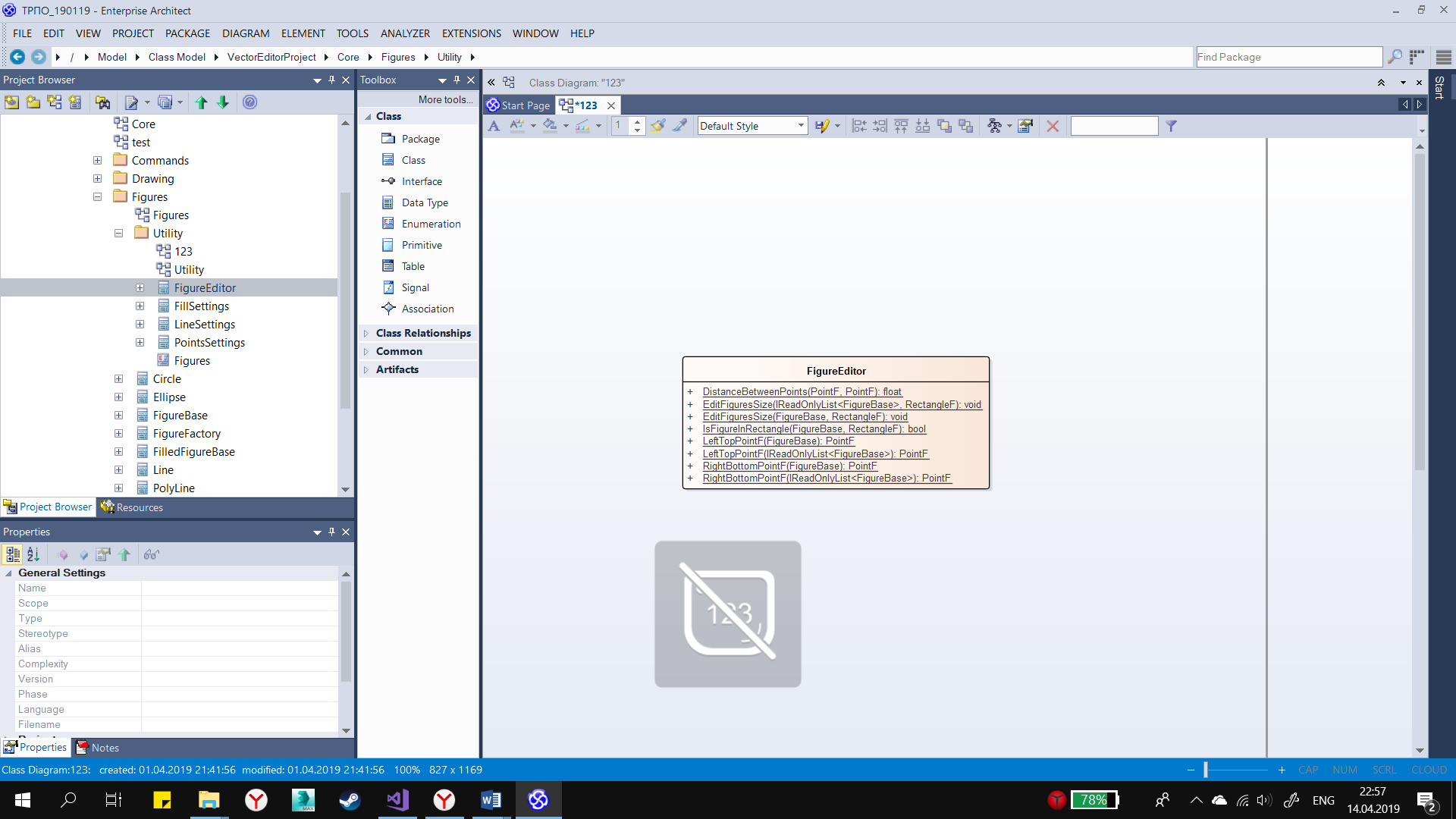


Рисунок 2.1 – Диаграмма класса FigureEditor

На данный момент класс FigureEditor упразднен, а его методы перенесены в те классы, которые располагают необходимыми данными, PointSettings, в основном.

## **2.2 Создатель**

Создатель (от англ. Creator). Класс должен создавать экземпляры тех классов, которые он может содержать или агрегировать, записывать, использовать или инициализировать, имея нужные данные.

Применение данного шаблона не повышает степень связности (Low Coupling), что является его преимуществом.

EditContext можно назвать положительным примером применения данного шаблона. Данный класс обладает нужной информацией для создания экземпляров состояний. Диаграмма классов состояний представлена на рисунке 2.2.

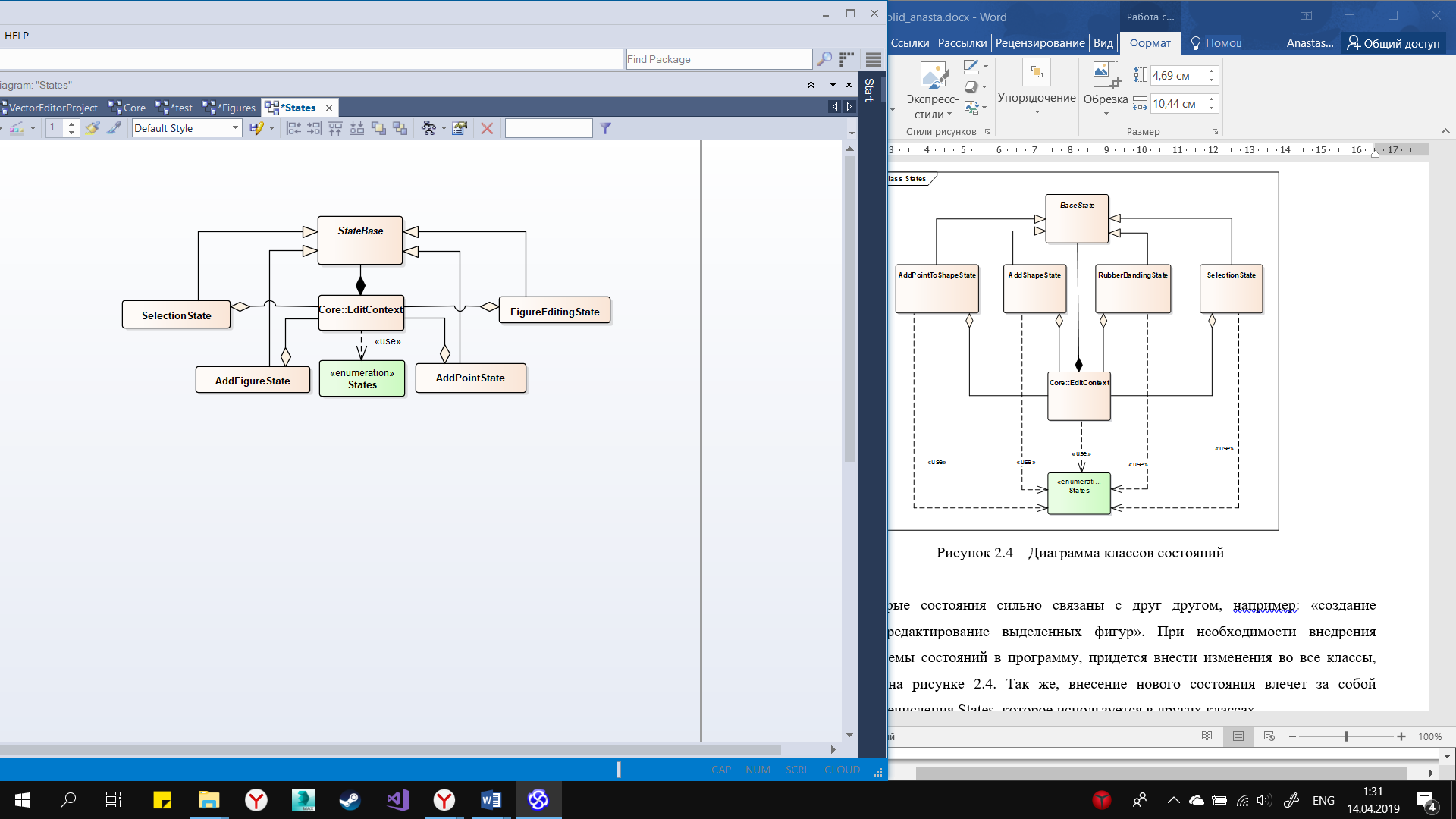


Рисунок 2.2 – Диаграмма классов состояний

В качестве отрицательного примера можно назвать класс Document, который по композируется и контролируется классом ControlUnit, однако иногда создается в DocumentForm. На рисунке 2.3 представлена соответствующая диаграмма классов.

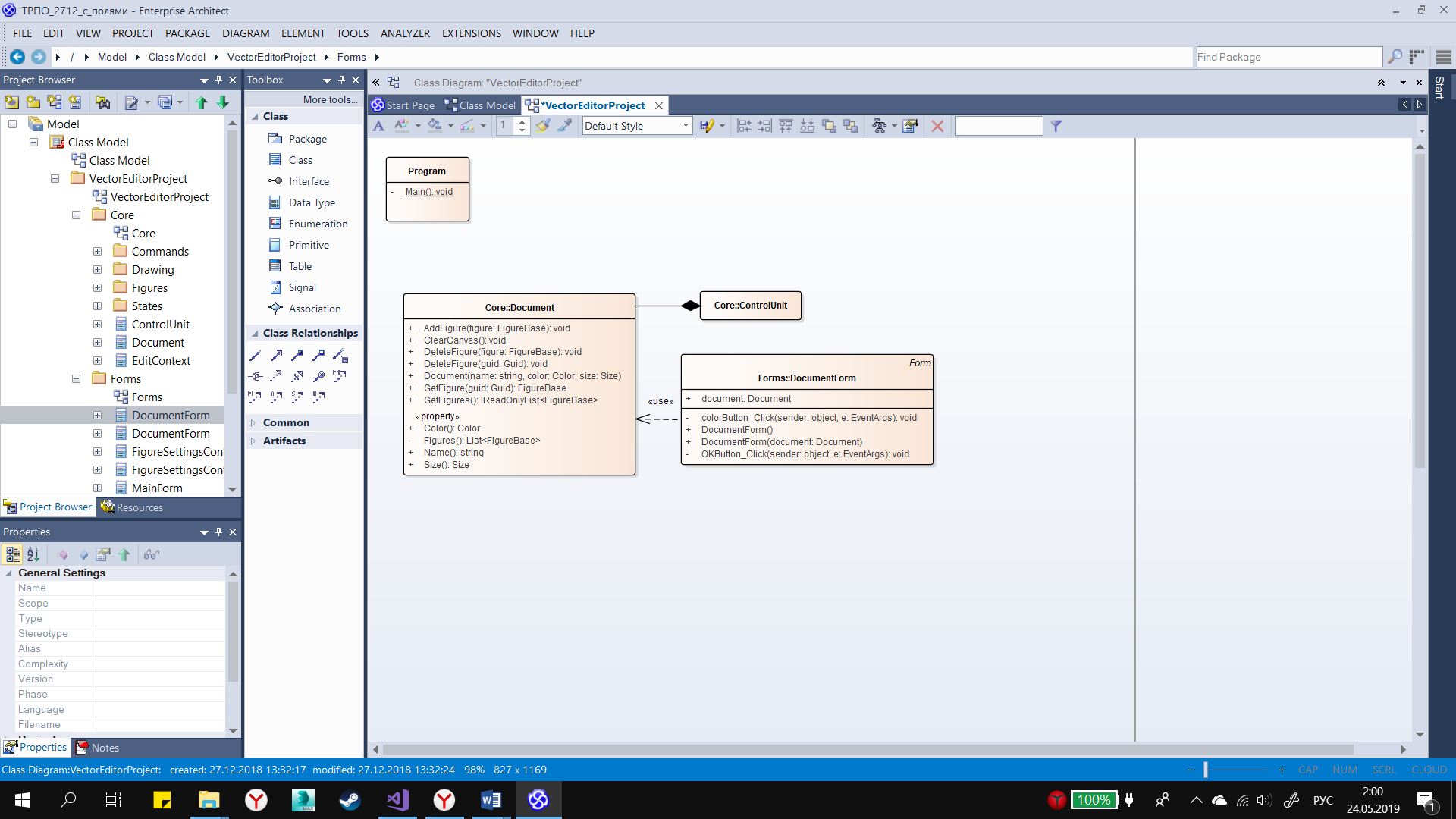


Рисунок 2.3 – Диаграмма класса Document

## **2.3 Контроллер**

Контроллер (от англ. Controller). Класс в данном случае отвечает за операции, запросы на которые приходят от пользователя, и может выполнять сценарии одного или нескольких вариантов использования (например, создание и удаление). Также класс не выполняет работу самостоятельно, а делегирует компетентным исполнителям.

Преимущества данного шаблона:

– удобно накапливать информацию о системных событиях;

– улучшаются условия для повторного использования компонентов.

Недостатком является то, что Контроллер может быть перегружен, что нарушает Information Expert и High Cohesion.

Класс EditContext можно привести в качестве положительного примера применения шаблона Контроллер. Он является менеджером, который принимает пользовательский ввод и делегирует его нужным состояниям: диаграмма классов состояний представлена на рисунке 2.2. На рисунке 2.4 представлен абстрактный класс состояния, которым EditContext делегирует пользовательский ввод, например, MouseUp, MouseDown.

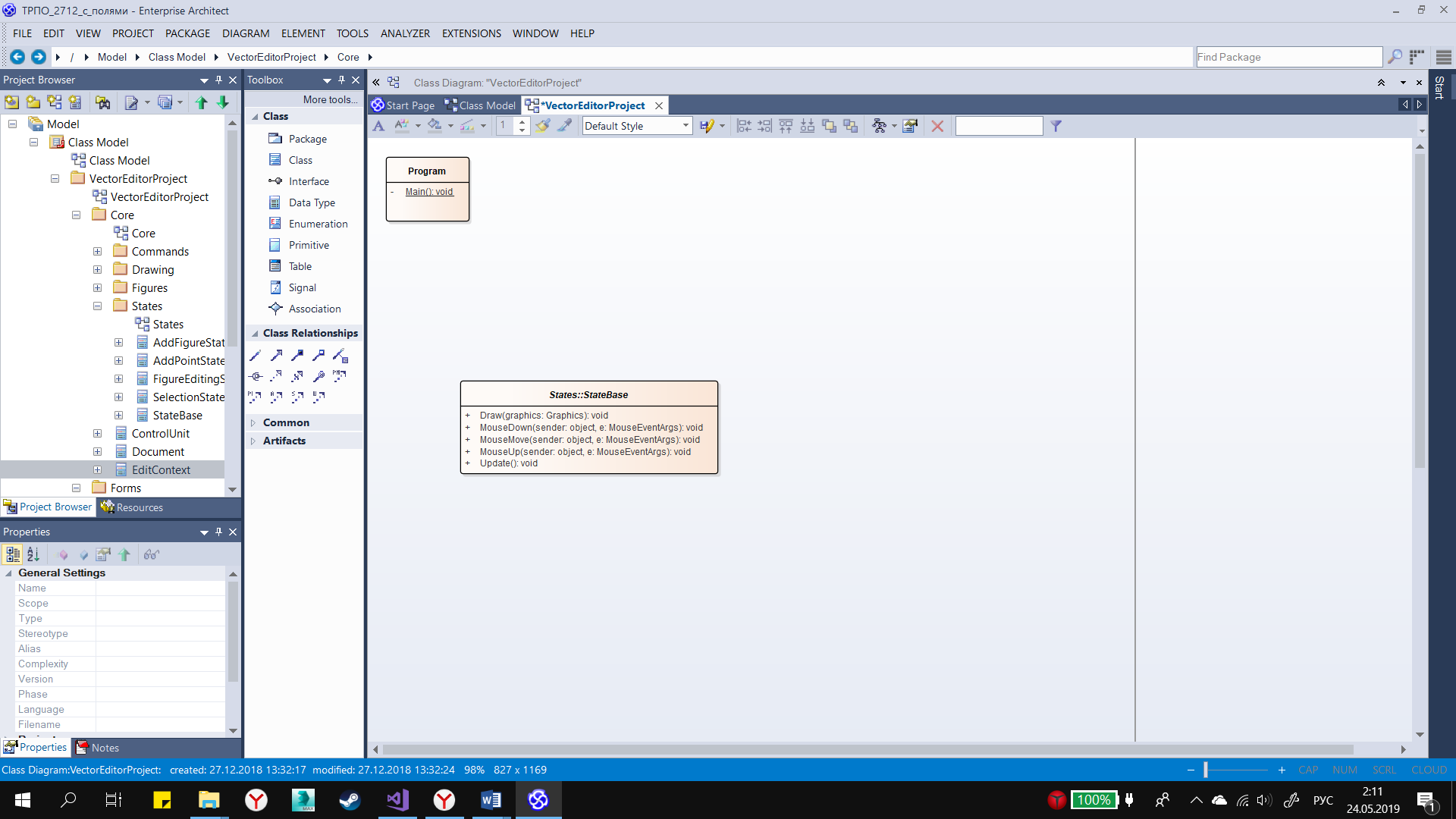


Рисунок 2.4 – Диаграмма класса StateBase

Изначально ControlUnit, диаграмма которого представлена на рисунке 2.5, задумывался как класс контроллер к стэку команд для Undo/Redo, но в итоге оказался перегружен обязанностями, и на данный момент полностью или частично отвечает за следующее:

– Undo/Redo;

– обновление .NET контролов после исполнения команд;

– удаление, копирование, вырезание фигур;

– сериализация/десериализация.

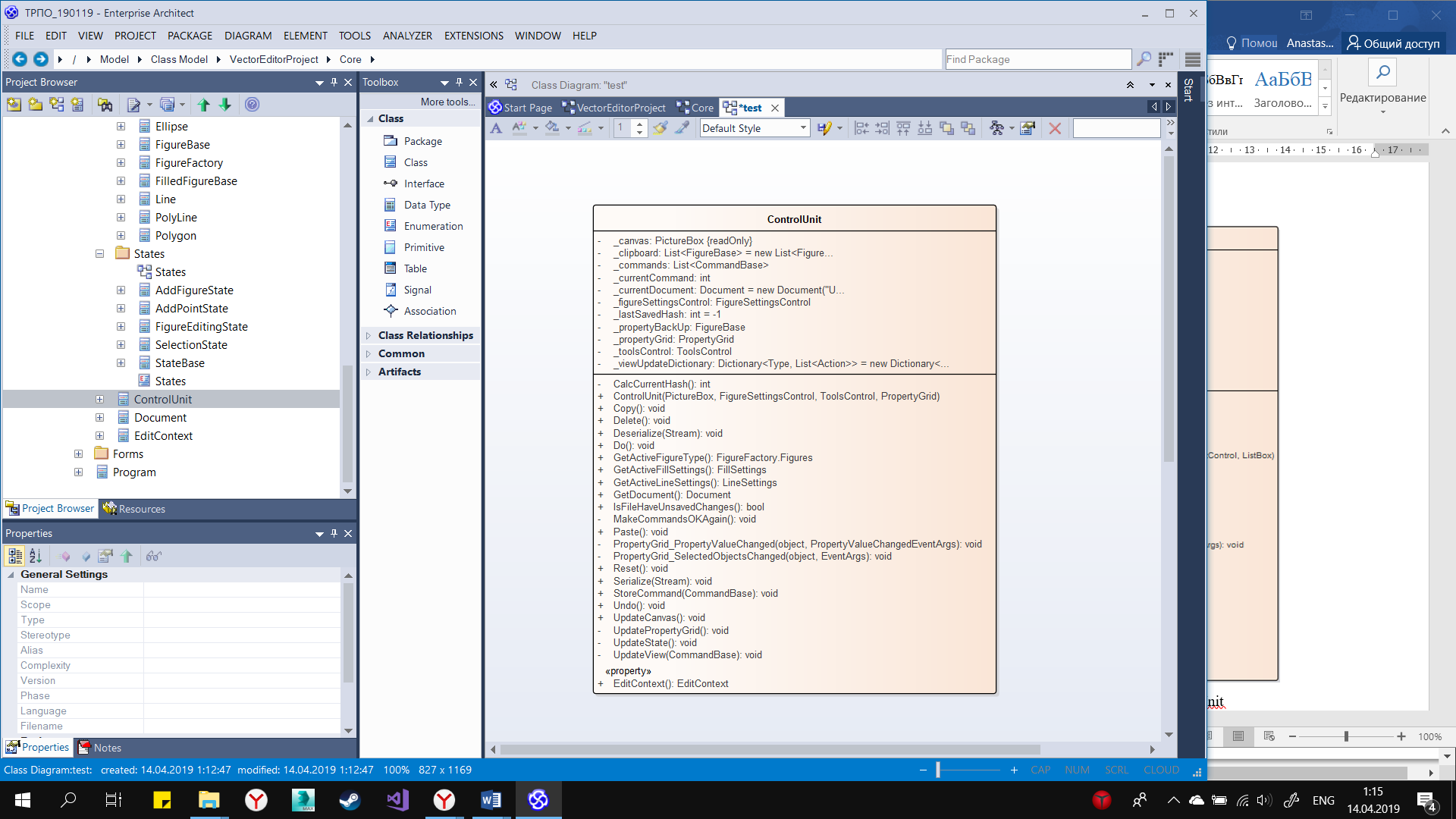


Рисунок 2.5 – Диаграмма класса ControlUnit

## **2.4 Полиморфизм**

Полиморфизм (от англ. Polymorphism). Устройство и поведение системы:

– определяется данными;

– задано полиморфными операциями ее интерфейса.

То есть, если поведение объектов одного типа (класс) может изменяться, обязанности распределяются для различных вариантов поведения с использованием полиморфных операций для этого класса.

Преимуществами данного шаблона являются возможности:

– легкого расширения и модернизации системы;

– введения новых реализаций без модификации клиентской части приложения (DI).

Недостатком является создание множественных интерфейсов.

Реализованное в программе взаимодействие с объектами фигур через полиморфные методы абстрактных классов является положительным примером применения данного шаблона. Изначально фигура к пользователю уже попадает с базовым типом от фабрики. Диаграмма класса FigureFactory представлена на рисунке 2.6. Эта часть будет выделена в SDK в лабораторной работе №4.

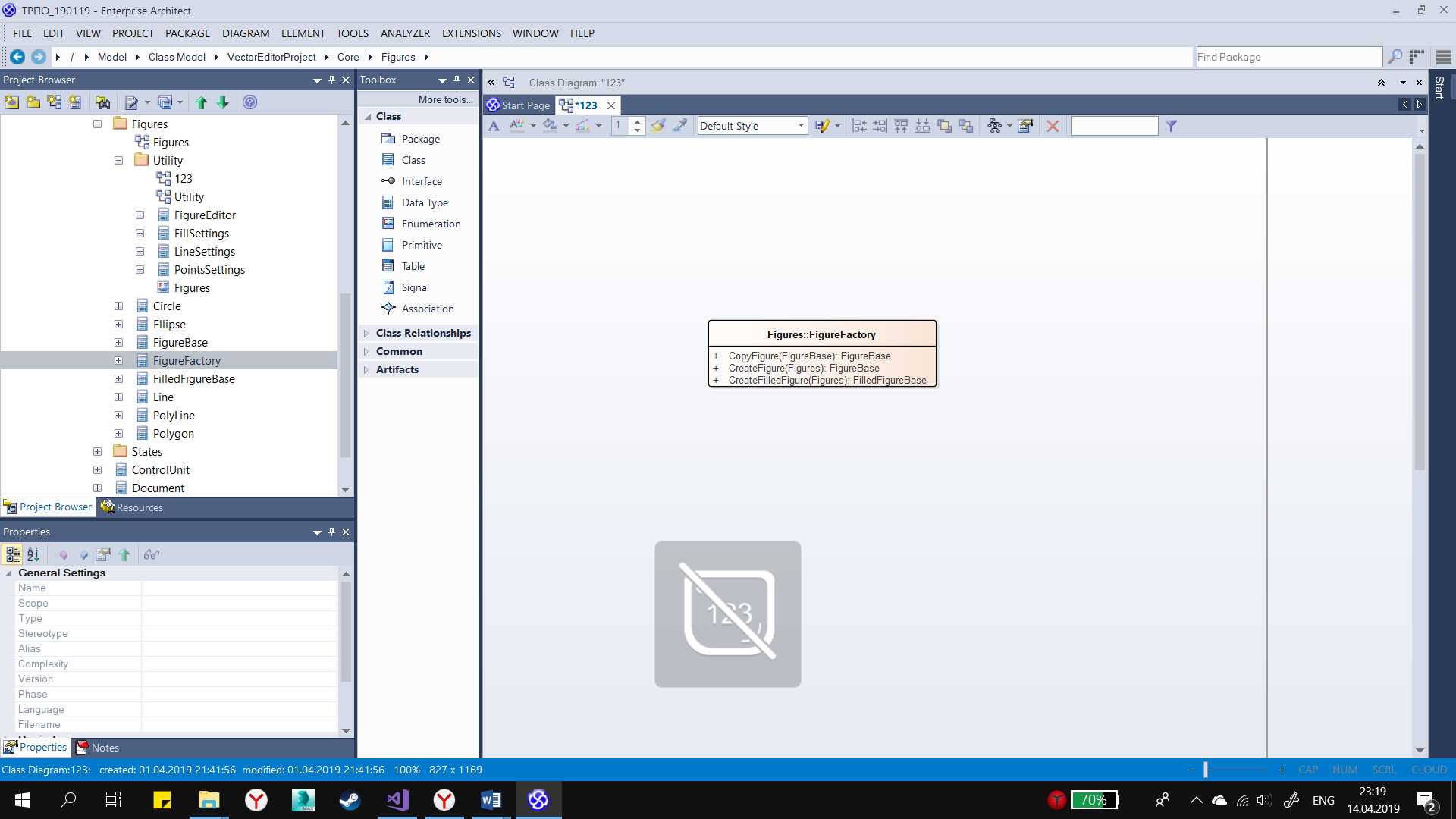


Рисунок 2.6 – Диаграмма класса FigureFactory

Жирный и выделяющийся отрицательный пример найти не удалось. В программе все конкретные реализации наследуются от абстракции и все взаимодействия с такими объектами ведутся через полиморфные методы.

## **2.5 Чистая выдумка**

Чистая выдумка (от англ. Pure Fabrication). Не относится к предметной области, но данный шаблон:

– уменьшает зацепление;

– повышает связанность;

– упрощает повторное использование.

Преимуществом данного шаблона является выполнение паттернов «Высокое Зацепление» и «Низкая Связность»: соответственно классы просты в поддержке и в повторном использовании.

Чистой выдумкой можно считать класс DrawerBase, диаграмма которого представлена на рисунке 2.7, и его наследников. Данная сущность абстрактна и неизвестна пользователю в рамках предметной области. Она создана только для того, чтобы не вешать обязанность отрисовки на сами фигуры.

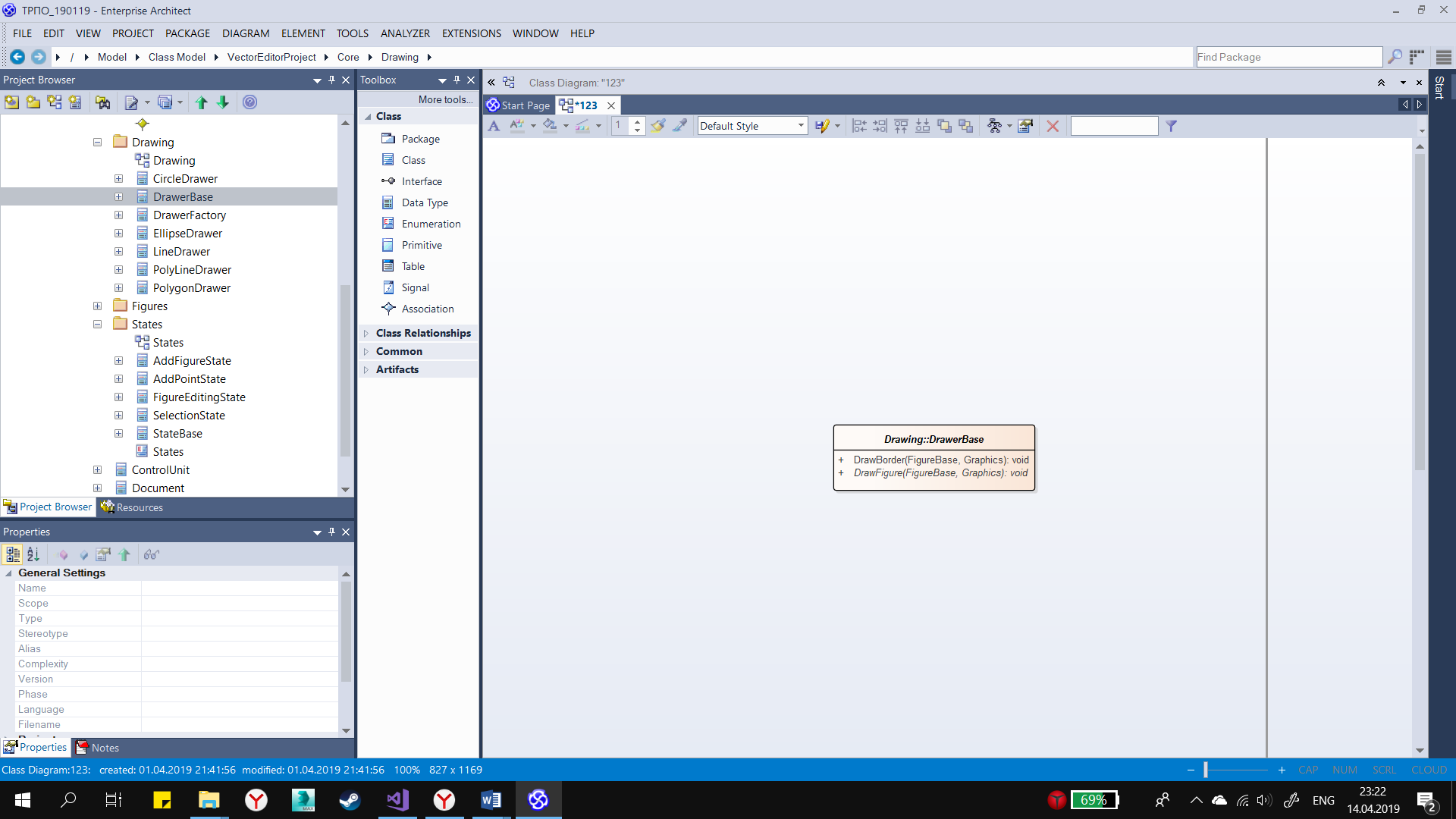


Рисунок 2.7 – Диаграмма класса DrawerBase

## **2.6 Посредник**

Посредник (от англ. Indirection). Слабое зацепление между элементами системы (и возможность повторного использования) обеспечивается назначением промежуточного объекта их посредником.

DrawerBase, диаграмма которого представлена на рисунке 2.7, является положительным примером применения шаблона Посредник. Данный класс создан, чтобы забрать у фигуры обязанность себя отрисовывать. На рисунке 2.8 представлена диаграмма классов, демонстрирующая систему отрисовки фигур.

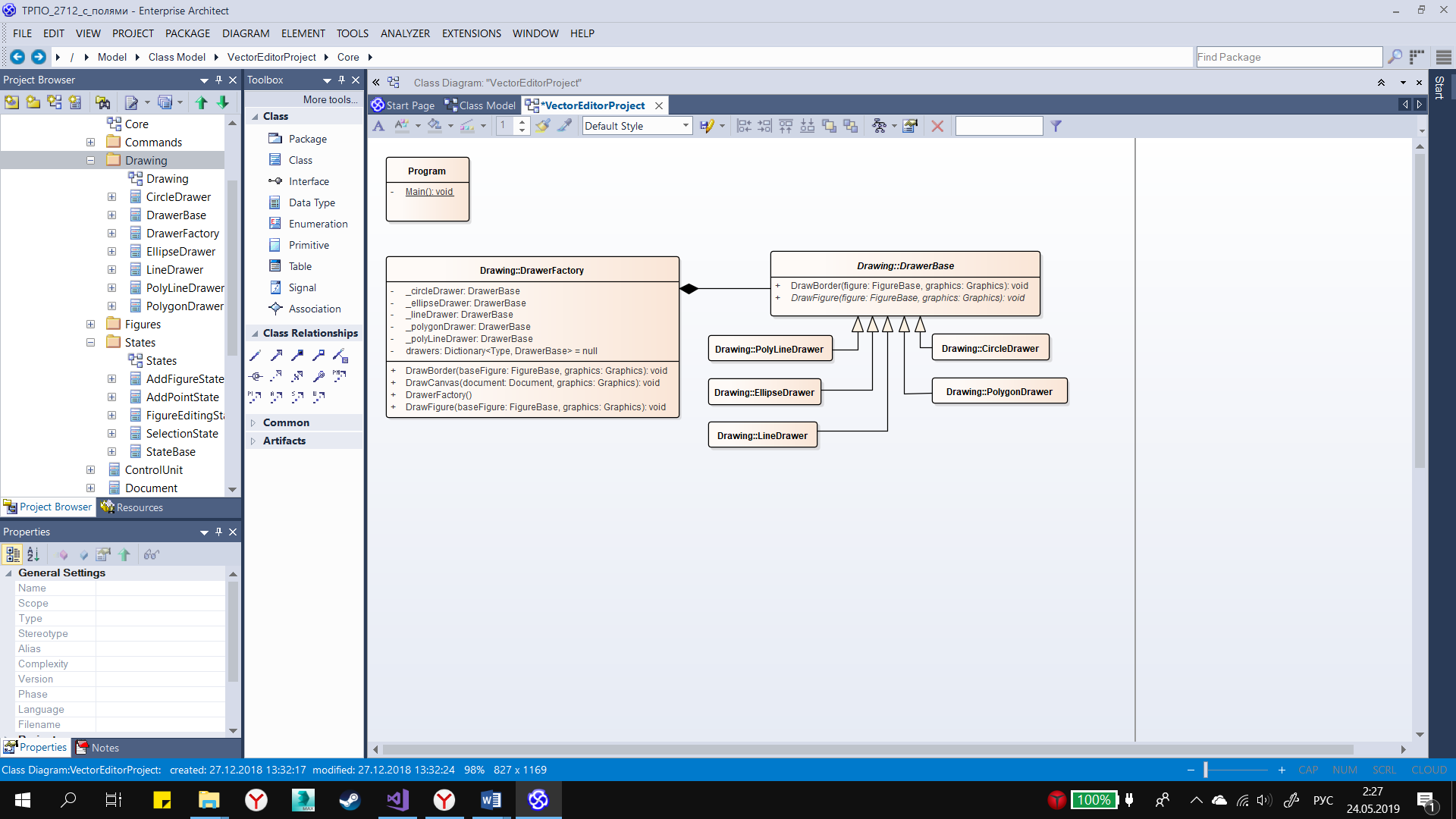


Рисунок 2.8 – Диаграмма классов

## **2.7 Сокрытие реализации**

Сокрытие реализации (также: Устойчивость к изменениям) (от англ. Protected Variations). Шаблон защищает элементы от изменения другими элементами (объектами или подсистемами) с помощью вынесения взаимодействия в фиксированный интерфейс, через который (и только через который) возможно взаимодействие между элементами. Поведение может варьироваться лишь через создание другой реализации интерфейса.

В описываемой программе данный шаблон проектирования не был использован.

# **3 Заключение**

В ходе работы был произведен анализ программного кода на соответствие паттернам GRASP, а именно оставшимся семи шаблонам, помимо Низкой связности (от англ. Low Coupling) и Высокого зацепления (от англ. High Cohesion), которые были рассмотрены в прошлой лабораторной работе.

В работе рассмотрены как положительные примеры применения шаблонов проектирования GRASP, так и негативные. В качестве примеров приведены образцы рассмотренного программного кода, а также соответствующие им UML-диаграммы.