Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

Высшего профессионального обучения

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

Отчет по лабораторной работе №2 по дисциплине

«Технология разработки программного обеспечения»

«Анализ на соответствие GRASP»

Выполнил:

Студент гр. 588-М1

\_\_\_\_\_\_Д.А. Домаскин

. «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

Проверил:

Доцент каф. КСУП

\_\_\_\_\_\_\_\_\_ А.А. Калентьев

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

Томск 2019

**Оглавление**

[**Введение** 3](#_Toc6863659)

[**1 Связность по содержимому** 4](#_Toc6863660)

[**2 Связность по внешним ссылкам** 5](#_Toc6863661)

[**3 Связанность по управлению** 6](#_Toc6863662)

[**4 Связность по образцу** 7](#_Toc6863663)

[**5 Связность по параметрам** 8](#_Toc6863664)

[**6 Связность по сообщениям** 9](#_Toc6863665)

[**7 Логическое зацепление** 10](#_Toc6863666)

[**8 Временное зацепление** 11](#_Toc6863667)

[**9 Процедурное зацепление** 12](#_Toc6863668)

[**10 Коммуникационное/информационное зацепление** 13](#_Toc6863669)

[**11 Последовательное зацепление** 14](#_Toc6863670)

[**12 Функциональное зацепление** 15](#_Toc6863671)

[**13 Зацепление по данным** 16](#_Toc6863672)

[**Заключение** 17](#_Toc6863673)

# **Введение**

GRASP - (англ. General Responsibility Assignment SoftwareиPatterns — общие образцы распределения обязанностей)— паттерны, используемые в ООП для решения общих задач по назначению обязанностей классам и объектам.

В данной лабораторной работе будет проведен анализ программного кода на соответствие принципам GRASP по связности и зацеплению.

**Типы связности**

* Content coupling (связность по содержимому) \*высокая
* Common coupling (связность по глобальным данным)
* External coupling (связность по внешним ссылкам)
* Control coupling (связность по управлению)
* Stamp coupling (Data-structured coupling) (связность по образцу)
* Data coupling (связность по параметрам)
* Message coupling (связность по сообщениям) \*низкая
* No coupling (отсутствует связность)

**Типы зацепления**

* Coincidental cohesion (случайное зацепление) \*худшее
* Logical cohesion (логическое зацепление)
* Temporal cohesion (временное зацепление)
* Procedural cohesion (процедурное зацепление)
* Communicational/informational cohesion
* (коммуникационное/информационное зацепление)
* Sequential cohesion (последовательное зацепление)
* Functional cohesion (функциональное зацепление) \*лучшее
* • Data cohesion (зацепление по данным)

.

# **1 Связность по содержимому**

Метод *UpMouseProcess* класса *DrawManager* и *PainterState* приведены на рисунке 1.1.

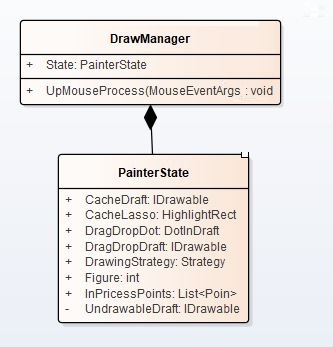


Рисунок 1.1 – Класс *DrawManager* и *PainterState*

Пример изменения локальных данных *PainterState* внутри метода класса  *DrawManager* приведен на листинге 1.1*.*

private void UpMouseProcess(MouseEventArgs e)

{

switch (State.DrawingStrategy)

{

case Strategy.TwoPoint:

State.InProcessPoints.Add(e.Location);

DraftPainter.AddToStorage();

break;

case Strategy.Multipoint:

{

switch (e.Button)

{

case MouseButtons.Left:

State.InProcessPoints.Add(e.Location);

DraftPainter.AddPointToCacheDraft(e.Location);

break;

case MouseButtons.Right:

DraftPainter.AddToStorage();

State.Figure = State.Figure;

break;

}

}

///...

Листинг 1.1 – Метод *UpMouseProcess*

# **2 Связность по внешним ссылкам**

Связность по внешним ссылкам появляется, когда два модуля разделяют:

• общий формат данных, описанный в ином месте

• протокол обмена данными

• интерфейс устройства.

В основном это связано с связи с внешними инструментами и устройствами.

В качестве примера можно привести связность GUI с экземпляром класса *DrawManager*, которая осуществляется с помощью ключей описанных в виде перечислений *Figure*. Код приведен ниже.

private void lineButton\_Click(object sender, EventArgs e)

{

\_drawManager.State.Figure = Figure.Line;

}

public enum Figure

{

Circle,

Ellipse,

Line,

Polyline,

Polygon

}

# **3 Связанность по управлению**

Диаграмма классов *DraftPainter* и *PainterState* приведена на рисунке 3.1.

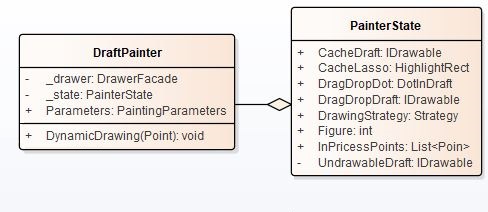


Рисунок 3.1 – Связь классов *DraftPainter* и *PainterState*

На листинге 3.1 показан пример связанности по управлению, где работа метода *DynamicDrawing* класса *DraftPainter* зависит от внутреннего состояния класса *PainterState*.

/// <summary>

/// Динамическая отрисовка

/// </summary>

/// <param name="mousePoint">Координаты мыши</param>

public void DynamicDrawing(Point mousePoint)

{

///...

switch (State.DrawingStrategy)

{

case Strategy.TwoPoint:

DoublePointDynamicDrawing(mousePoint);

break;

case Strategy.Multipoint:

MultiPointDynamicDrawing(mousePoint);

break;

case Strategy.Selection:

LassoDynamicDrawing(mousePoint);

break;

}

}

Листинг 3.1 – Код метода *DynamicDrawing*

# **4 Связность по образцу**

В качестве примера связности по образцу рассмотрим метод *PullPoints* класса *StorageManager*. Ниже приведен код метода *PullPoints*, а на рисунке 4.1 диаграмма интерфейса *IDrawable*.

/// <summary>

/// Получить точки из фигуры

/// </summary>

/// <param name="item">Фигура, из которой нужно вытащить точки</param>

/// <returns>Точки фигуры</returns>

public List<Point> PullPoints(IDrawable item)

{

var pullPointList = new List<Point>();

if (item is IMultipoint multipoint)

{

foreach (var pointInDraft in multipoint.DotList)

{

pullPointList.Add(pointInDraft);

}

}

else

{

pullPointList.Add(item.StartPoint);

pullPointList.Add(item.EndPoint);

}

return pullPointList;

}

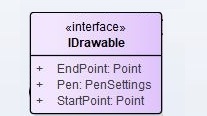


Рисунок 4.1 – Диаграмма интерфейса *IDrawable*

# **5 Связность по параметрам**

На рисунке 5.1 приведена диаграмма классов *DrawManager* и *Selector*.

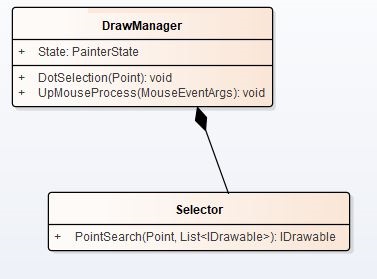


Рисунок 5.1 – Связь классов *DrawManager* и *Selector*

На листинге 5.1 приведен пример связанности по параметрам.

private void DotSelection(Point mousePoint)

{

DraftStorageManager.DiscardAll();

var selectedDraft = \_selector.PointSearch(

mousePoint,

DraftStorageManager.PaintedDraftStorage);

if (selectedDraft != null)

{

DraftStorageManager.EditHighlightDraft(selectedDraft);

}

}

Листинг 5.1 – Код метода *DotSelection*

# **6 Связность по сообщениям**

На рисунке 6.1 приведена диаграмма связанных классов *GraphicsForm* и *SelectionPanel*.

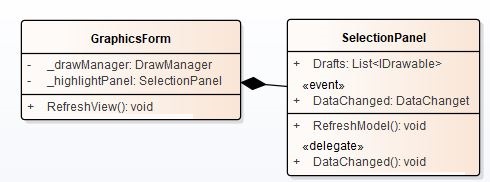


Рисунок 6.1 – Связь классов *GraphicsForm* и *SelectionPanel*

private void RefreshModel()

{

///...

ModelChanged();

///...

}

Листинг 6.1 – Фрагмент метода *RefreshModel* класса *SelectionPanel*

На листинге 6.2 приведен фрагмент конструктора класса *GraphicsForm*.

public GraphicsForm()

{

///...

\_highlightPanel = new SelectionPanel();

\_highlightPanel.ModelChanged += \_drawManager.DraftPainter.RefreshCanvas;

///...

}

Листинг 6.2 – Фрагмент конструктора класса *GraphicsForm*

# **7 Логическое зацепление**

В качестве логического зацепления рассмотрим методы класса *Selector* рисунок 7.1.

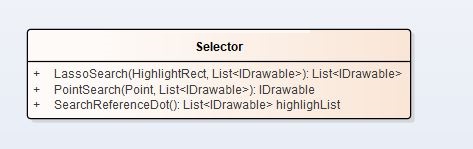


Рисунок 7.1 – Диаграмма класса *Selector*

Методы данного класса выполняют не связаны ни по данным, ни по управлению, но логически связаны тем, что выполняют поиск объектов.

# **8 Временное зацепление**

Временное зацепление имеет место, если элементы объединены вместе, потому что должны использоваться примерно одновременно. Так как на данном этапе в разрабатываемом графическом редакторе отсутствуют алгоритмы, в которых есть зависимость от времени, логично, что примеров временного зацепления найдено не было.

# **9 Процедурное зацепление**

В качестве процедурного зацепления рассмотрим метод *Cut* класса *DrawManager* листинг 9.1.

/// <summary>

/// Вырезать объект

/// </summary>

/// <param name="buffer">Буфер обмена</param>

public void Cut(DraftClipboard buffer)

{

buffer.SetRange(DraftStorageManager.HighlightDraftStorage);

DraftStorageManager.RemoveRangeHighlightDrafts();

DraftPainter.RefreshCanvas();

}

Листинг 9.1 – Фрагмент конструктора класса *DrawManager*

Данный метод *Cut* последовательно вызывает методы других классов в нужной последовательности.

# **10 Коммуникационное/информационное зацепление**

В качестве примера коммуникационного/информационного зацепления рассмотрим класс *StorageManager*, его диаграмма приведена на рисунке 10.1.

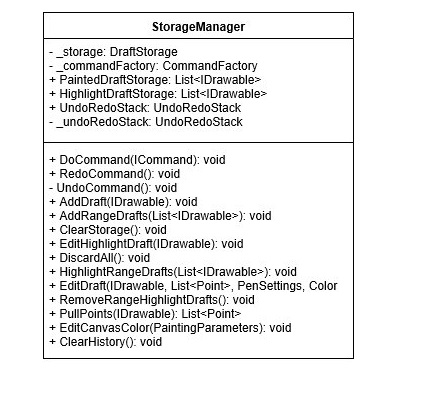


Рисунок 10.1 – Диаграмма класса *StorageManager*

Методы данного класса выполняют разноплановые операции, связанные тем, что производят изменения состояние объекта хранилища *\_storage: DraftStorage*.

# **11 Последовательное зацепление**

В качестве последовательного зацепления рассмотрим конструктор класс *DrawManager*, код представлен на рисунке 11.1.

public DrawManager(Graphics \_paintCore)

{

DraftPainter = new DraftPainter(\_paintCore);

DraftStorageManager = new StorageManager(new DraftStorage());

State = new PainterState();

DraftPainter.State = State;

DraftPainter.Corrector = DraftStorageManager;

\_selector = new Selector();

}

Листинг 11.1 – Фрагмент конструктора класса *DrawManager*

Данный код последовательно инициализирует объекты, которые являются необходимыми параметрами для инициализации других.

# **12 Функциональное зацепление**

Пример функционального зацепления приведен в коде класса *DraftSerealizer* на листинге 12.1.

public class DraftSerealizer

{

/// <summary>

/// Сериализовать стек команд

/// </summary>

/// <param name="stream">Поток</param>

/// <param name="stack">Сериализуемый стек</param>

public void Serialize(Stream stream, UndoRedoStack stack)

{

BinaryFormatter binaryFormatter = new BinaryFormatter();

binaryFormatter.Serialize(stream, stack);

}

/// <summary>

/// Десериализовать стек комманд

/// </summary>

/// <param name="stream">Поток</param>

/// <returns>Выполненные комманды</returns>

public UndoRedoStack Deserialize(Stream stream)

{

BinaryFormatter binaryFormatter = new BinaryFormatter();

return (UndoRedoStack)binaryFormatter.Deserialize(stream);

}

}

Листинг 12.1 – Фрагмент конструктора класса *DrawManager*

# **13 Зацепление по данным**

В качестве примера зацепления по данным можно привести класс *DraftClipboard*, его код приведен на листинге 13.1.

public class DraftClipboard

{

/// <summary>

/// Хранилище объектов буфера обмена

/// </summary>

private List<IDrawable> \_clipboard = new List<IDrawable>();

/// <summary>

/// Фабрика фигур для клонирования

/// </summary>

private DraftFactory \_factory = new DraftFactory();

/// <summary>

/// Записать в буфер ряд объектов

/// </summary>

/// <param name="items">Записываемые объекты</param>

public void SetRange(List<IDrawable> items)

{

\_clipboard.Clear();

foreach(var item in items)

{

\_clipboard.Add(\_factory.Clone(item));

}

}

/// <summary>

/// Вернуть из буфера ряд объектов

/// </summary>

/// <returns></returns>

public List<IDrawable> GetAll()

{

var returnList = new List<IDrawable>();

foreach (var item in \_clipboard)

{

returnList.Add(\_factory.Clone(item));

}

return returnList;

}

}

Листинг 13.1 – Код класса *DraftClipboard*

# **Заключение**

В ходе выполнения лабораторной работы был проанализирован программный код на соответсвие принципам GRASP по типам связностей и зацеплений. Полученные результаты анализа помогут для дальнейшей разработки приложения, так как выявляют слабые участки кода, на которые нужно обратить особое внимание.