**УДК 004.6**

**ДАТА-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ: ПОЛОЖИТЕЛЬНЫЕ И ОТРИЦАТЕЛЬНЫЕ АСПЕКТЫ**

***А.А. Файтельсон***

*Бакалавр второго года обучения по направлению подготовки «Математическое обеспечение и администрирование информационных систем»*

*Курский государственный университет*

*e-mail: z0tedd@gmail.com*

*Научный руководитель:*

***А.В. Кривонос***

*Кандидат технических наук, доцент кафедры программного обеспечения и администрирования информационных систем*

*Курский государственный университет*

*e-mail: krivonos\_av@kursksu.ru*

*Статья посвящена рассмотрению дата-ориентированного программирования, положительных и отрицательных аспектов этой парадигмы. Данная парадигма призвана помочь в проектировании информационных систем*

***Ключевые слова:*** *дата-ориентированное программирование, парадигмы программирования, проектирование информационных систем*

**Введение**. «Кризис программного обеспечения» был впервые выявлен в 1968 году [1] и в прошедшие десятилетия он скорее углубился, чем ослаб. Самая большая проблема в разработке и сопровождении крупномасштабных программных систем - сложность, большие системы трудно понять, а следовательно, спроектировать и реализовать.

В своей классической статье «Серебряной пули нет» Брукс [2] выделил четыре особенности программных систем, которые усложняют создание программного обеспечения: сложность, согласованность, изменчивость и невидимость. Из них считается, что сложность является более значимым признаком; остальные можно классифицировать как формы сложности или рассматриваться как проблематичные исключительно из-за сложности запутанность в системе.

Сложность является основной причиной подавляющего большинства проблем с программным обеспечением. Ненадежность, отсутствие безопасности, зачастую даже плохую производительность в крупномасштабных системах можно рассматривать как результат, в конечном итоге, неуправляемой сложности.

Актуальность проблемы сложности восприятия систем широко признана. Как сказал Дейкстра [3]:

«...Мы должны делать системы четкими, распутанными и простыми, если не хотим быть раздавленными сложностью, которую мы сами создали...»

Первичный статус сложности как основная причина других проблем заключается просто в том факте, что способность понять систему является предпосылкой для того, чтобы избежать всех проблем. [4] Для решения проблемы понимания системы можно использовать парадигму дата-ориентированного программирования.

**Дата-ориентированное программирование** (ДОП, Data-Oriented Programming) — подход к разработке информационных систем, в котором данные занимают центральное место.

Основным элементом функционального программирования являются результаты выполнения функций, а объектно-ориентированное программирование (ООП) в основном имеет дело с объектами. В обоих случаях в центр ставится код: в одном случае это некоторый результат, выполнение кода, в другом — сгруппированный код, связанный с неким внутренним состоянием.

ДОП смещает фокус внимания с объектов на сами данные: тип данных, их расположение в памяти, способы их считывания и обработки в приложения. В отличие от традиционного ООП, где данные скрыты внутри объектов и смешаны с логикой, в ДОП основное внимание уделяется отделению данных от логики.

Принципы дата-ориентированного программирования [5]:

1. отделение поведения от данных;
2. представление данных в общем виде;
3. отделение схемы от представления;
4. данные неизменны.

Принцип №1 — это принцип проектирования, который рекомендует четкое разделение кода(поведение) и данных. Может показаться, что это принцип функционального программирования (ФП), но в действительности придерживаться его или нет можно, как и в функциональном программировании, так и в объектно-ориентированном.

Следование этому принципу в ООП означает агрегирование кода как методов статического класса в ООП языках. Нарушение этого принципа в ФП означает сокрытие состояния программы в лексической области видимости функции.

При соблюдении принципа №1 код отделяется от данных. ДОП не зациклен на структурах, которые следует использовать для организации кода, но в нем большое внимание уделяется тому, как должны быть представлены данные. В этом заключается принцип №2 – представление данных в общем виде.

В ДОП данные представлены с помощью общих структур данных словарей или массивов вместо создания экземпляров данных с помощью определенных классов. Фактически, большинство сущностей можно представить в качестве словарей и массивов, но можно использовать и другие общие структуры данных (деревья, очереди, множества).

Принцип №3 пропагандирует то, что данные должны быть неизменяемыми. ДОП очень строг в этом вопросе. Мутация данных не допускается! В ДОП изменения данных выполняются путем создания новых версий данных. Ссылка на переменную может быть изменена так, чтобы она ссылалась на новую версию данных, но само значение данных никогда не должно меняться.

С данными, отделенными от кода и представленными с помощью общих и неизменяемых структур данных, теперь возникает вопрос, как выразить форму данных. Принцип №4 отвечает на данный вопрос. В ДОП ожидаемая форма выражается как схема данных, которая хранится отдельно от них. Главное преимущество этого принципа заключается в том, что он позволяет разработчикам решать, какие фрагменты данных должны иметь схему, а какие - нет.

**Положительные аспекты Дата-ориентированного программирования**. Кратко изложенные принципы данной парадигмы помогут понять плюсы от ее использования. Из первого принципа вытекает, что тщательное разделение кода и данных приносит пользу нашим программам следующим образом:

1. код можно повторно использовать в разных контекстах;
2. код можно тестировать изолированно;
3. системы, как правило, менее сложны.

Когда код не отделен от данных, чтобы протестировать метод необходимо создать объект, содержащий его. В простом сценарии это не представляет большой сложности. Можно проинициализировать объект со всеми его методами и данными, но в реальности очень часто приходится подгружать некоторые данные из вне, из-за чего создание объекта для тестирование становится затруднительной задачей. В то время как при использовании дата-ориентированной парадигмы, создание объекта, содержащего тестируемые методы, в разы проще, потому что не происходит получение труднодоступных данных.

Использование универсальных структур данных для представления данных имеет множество преимуществ, такие как возможность использования универсальных функций, которые не ограничиваются нашим конкретным вариантом использования. Во многих языках программирования существует огромный набор функций для работы с универсальными структурами данных, к тому же этот набор расширяется дополнительными библиотеками. Также данный принцип позволяет строить информационные системы с более гибкой моделью данных. Данные не привязываются к определенной форме, они могут быть созданы без предопределенной формы, а их форма может быть изменена по желанию.

В классическом ООП каждый фрагмент данных создается с помощью класса и должен соответствовать строгой форме. Когда требуется немного другая форма данных, необходимо определить новый класс, однако при использовании универсальных структур данных поля могут быть добавлены или удалены из словаря мгновенно.

Когда программы ограничены от мутации данных, мы получаем выгоду во многих отношениях, а именно:

1. доступ к данным без страха изменения;
2. предсказуемое поведение кода;
3. быстрые проверки равенства;
4. безопасность многопоточного доступа.

При создании многопоточных программ для избежания проблем формата «гонки данных» используются примитивы синхронизации потоков – мьютексы. Их использование ухудшает производительность, а также накладывает на программиста дополнительную нагрузку, что делает разработку сложнее. При использовании принципа №3 мы можем не волноваться о «гонке данных», потому что данные никогда не будут изменены, следовательно не будет возникать проблемы синхронизации потоков.

Разделение схемы данных от представления дает свободу выбора данных для проверки, а также позволяет создавать необязательные поля, расширенные условия проверки. Присутствие схемы данных, позволяет переводить ее в готовые диаграммы, что может послужить более простому пониманию, а следовательно, проектированию всей системы.

**Отрицательные аспекты дата-ориентированного программирования**. Цена, которую мы платим за получение выгоды от разделения кода и данных состоит из 3 пунктов:

1. не существует контроля над тем, какой код к каким данным имеет доступ;
2. нет упаковки данных и методов работы над ними в один класс;
3. система состоит из большего количества сущностей.

Когда код и данные смешаны, легко понять, к каким фрагментам кода можно получить доступ. Например, в ООП данные инкапсулируются в объект, который гарантирует, что данные доступны только методами объекта. В ДОП данные стоят сами по себе. Они прозрачные для всех участков кода. Когда происходит изменение формы данных при перепроектирование, нужно будет изменять методы, работающими с ними.

Так как код находится отдельно от данных, то методы работы с ними могут лежать в разных директориях и пакетах в связи с чем возникает проблема того, что разработчик может не знать о существовании того или иного фрагмента кода, из-за чего появляется не нужная дупликация функционала.

Проблемы, возникающие при представлении данных в общем виде таковы:

1. снижение производительности;
2. отсутствие структуры класса, а также проверок компилятора на соответствие аргументов в функциях работы с данными;
3. необходимость приведения типов в статически типизированных языках.

Под отсутствием структуры класса подразумевается, то, что у разработчика не будет некоторого класса со всеми полями и методами, а будет один обобщенный объект в виде словаря, для работы с которым будут отдельно доступны методы. В случае представления в виде обобщенного объекта подсказки среды разработки также будут недоступны.

Использование неизменных данных накладывает вычислительные ограничения, а также в некоторых языках требует сторонних библиотек.

В зависимости от реализации разделения данных и их вида, может повлечь за собой более слабую связь между двумя этими понятиями, а также появление дополнительных затрат на проверку данных в соответствии с их схемой.

**Вывод.** Несмотря на некоторые отрицательные аспекты, ДОП упрощает проектирование и реализацию информационных систем, ставя в приоритет представление данных. Достигается ДОП следованием 4 основным принципам:

1. отделение кода от данных;
2. представление данных приложения с помощью общих структур данных;
3. обработка данных как неизменяемых;
4. отделение схемы данных от представления данных.

Список используемой литературы:

1. Naur P., Randell B. SOFTWARE ENGINEERING // NATO SOFTWARE ENGINEERING CONFERENCE. Garmisch, 1968.
2. Brooks F.P.Jr The Mythical Man-Month: Essays on Software Engineering [Текст] / Brooks F.P.Jr — 2. — : Addison-Wesley, 1995 — 336 c.
3. Dijkstra E.W. The tide, not the waves / Dijkstra E.W. [Текст] // Beyond Calculation: The Next Fifty Years of Computing. — :Copernicus, 1997. — С. 59–64. Moseley B., Marks P.
4. Out of the Tar Pit / Moseley B., Marks P. [Текст] // Software Practice Advancement. — :, 2006. — С. .
5. Sharvit Yehonathan Data-Oriented Programming: Reduce software complexity [Текст] / Sharvit Yehonathan — 1. — : Manning Publications Co, 2022 — 426 c, ISBN: 9781617298578

GRNTI 20.53.24

**PAPER TITLE**

**A.A. Faitelson**

Bachelor of the second year of study in the direction of training "Software technology and administration of information systems"

Kursk State University

e-mail: z0tedd@gmail.com

Scientific supervisor:

**A.V. Krivonos**

PhD of Technical Sciences, Associate Professor of the Department of Software and Administration of Information Systems

Kursk State University

e-mail: krivonos\_av@kursksu.ru

*The article is devoted to the consideration of data-oriented programming, positive and negative aspects of this paradigm. This paradigm is intended to help in the design of information systems*

***Keywords****: data-oriented programming, programming paradigms, design of information systems*