О МАТЕМАТИЧЕСКОМ ОБОСНОВАНИИ SOLID ПРИНЦИПОВ

Е.А. Тюменцев

ОмГУ им. Ф.М. Достоевского, Мира 55-А, 644077 Омск, Россия
etyumentcev@gmail.com

SOLID – это аббревиатура, образованная названиями пяти архитектурных принципов объектно-ориентированного программирования: The **S**ingle Reposnsibility Principle (SRP), The **O**pen-Closed Principle (OCP), The **L**iskov Substitution Principle (LSP), The **I**nterface Segregation Principle (ISP), The **D**ependency Inversion Principle (DIP). Впервые LSP был рассказан Барбарой Лисков на коференции OOPSLA'87 [1], оставльные принципы опубликованы в книге Бертрана Мейера [2] в 1988 году. Значительную роль в популяризации SOLID сыграл Роберт Мартин, который опубликовал серию статей [3–6] в журнале The C++ Report. Он же придумал аббревиатуру SOLID.

Для удобства читателя приведем формулировки данных принципов:

The Single Responsibility Principle. *Должна быть ровно одна причина для изменения класса.*

The Open-Closed Principle. *Программные сущности (классы, модули, функции и т.п.) должны быть открыты для расширения, но закрыты для изменения.*

The Liskov Substitution Principle. Функции, которые используют ссылки на базовые классы, должны иметь возможность использовать объекты производных классов, не зная об этом.

The Interface Segregation Principle. *Клиенты не должны зависеть от методов, которые они не используют.*

The Dependency Inversion Priciple. *Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.*

Несмотря на то, что принципам уже почти три десятка лет, до сих пор не сложилось единого мнения относительно целесообразности их применения на практике, потому что привычные для программистов конструкции: оператор для создания нового объекта *new*, *enum*, оператор множественного выборa *switch*, цепочка вызовов *if–else–if*, операторы приведения типа и т.д, за редким исключением, противоречат SOLID.

Так, проблема со *switch* в том, что он требует явного перебора всех возможных вариантов, что на практике не всегда возможно. Например, в графическом редакторе в данной момент есть четрые графических примитива: линия, эллипс, прямоугольник, ломаная. Процедура отрисовки изображения использует *switch* по типу графического примитива для отрисовки каждого графического элемента рисунка. Нет никакой гарантии, что через некотрое время не потребуется добавить какой-нибудь новый элемент, скажем, встроенное изображение или распылитель. Тогда придется модифицировать *switch*, что прямо нарушает OCP.

Оператор *new* требует явного указания типа, который он создает. А это значит, что если потребуется создать объект другого типа, то придется модифицировать оператор *new*, что опять нарушает OCP. Чтобы избавиться от данного недостатка используется паттерн Абстрактная фабрика [7].

Возникает вопрос: можно ли формально доказать или опровергнуть SOLID?

Удалось получить математическое обоснование всех 5 принципов, используя логику Хоара [8]. Алфавитом в логике Хоара является так называемая тройка Хоара

$$\{P\}S\{Q\},$$

где P, Q – утверждения – формулы логики предикатов, P называют предусловием, Q – постусловием, S – команда какого-либо языка программирования.

В доказательстве задействованы только две аксиомы данной логики:
Аксиома композиции.

$$\{P\}S\{Q\}, \{Q\}T\{R\} \vdash \{P\}S;T\{R\}$$

Аксиома выводимости

$$P_1 \to P, \{P\}S\{Q\}, Q \to Q_1 \vdash \{P_1\}S\{Q_1\}$$

Считается, что SOLID — это принципы объектно-ориентированного программирования. Однако поскольку в доказательстве не делалось никаких предположений о структуре самих операторов, то SOLID принципы справедливы и для процедурного программирования. А поскольку не использовались аксиомы присваивания и цикла, то SOLID также справедливы и для функционального программирования.

Литература

1. Liskov B. Keynote address — data abstraction and hierarchy // ACM SIGPLAN Notices. 1988. Vol. 23. No. 5. P. 17–34.

2. Meyer B. Object-oriented Software Construction. Prentice Hall, New York 1988.

3. Martin R. The Open-Closed Principle // C++ Report. January 1996.

4. Martin R. The Liskov Substitution Principle // C++ Report. March 1996.

5. Martin R. The Dependency Inversion Principle // C++ Report. May 1996.

6. Martin R. The Interface Segregation Principle // C++ Report. June 1996.

7. Gamma E. Helm R. Larman C. Johnson R. Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, Limited, 2005.

8. Hoare C. A. R. An axiomatic basis for computer programming // Magazine Communications of the ACM. 1969. Vol. 12. No. 10. P. 576–580.