Математическое и компьютерное моделирование:
материалы III Международной научной конференции
(Омск, 12 ноября 2015 г.). Омск, 2015. С. 165–167. УДК 004.4

Е.А. Тюменцев

Омский государственный университет им. Ф.М. Достоевского, г. Омск

SOLID ПРИНЦИПЫ И ЛОГИКА ХОАРА

SOLID – это акроним, полученный из первых букв в названии пяти архитектурных принципов объектно-ориентированного программирования:

Single Reposnsibility Principle (SRP),

Open-Closed Principle (OCP),

Liskov Substitution Principle (LSP),

Interface Segregation Principle (ISP),

Dependency Inversion Principle (DIP).

Принцип подстановки Лисков (LSP) назван в честь своего автора – Барбары Лисков, которая впервые рассказала о нем на конференции OOPSLA'87 [1]. Четыре остальных опубликованы впервые в книге Бертрана Мейера [2] в 1988 году. SOLID стали известными благодаря статьям Роберта Мартина, опубликовавшего серию статей [3–6] в журнале The C++ Report и книгу [7]. Р. Мартин так же является автором самого акронима.

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.*

SOLID принципы обладают рядом интересных следствий, которые делают вопрос их обоснованности и применимости очень важным.

Например, конструкции языков программирования, такие как, оператор множественного выбора *switch,* перечислимый тип *enum,* цепочка вложенных операторов *if-else-if* подразумевают перебор нескольких вариантов. Если заранее не удается выявить полный список всех вариантов, то рано или поздно приходится вносить изменения в соотвествующую конструкцию, а это нарушает OCP. На практике, приходится иметь дело как раз с ситуациями, когда весь набор вариантов заранее неизвестен, либо меняется с течением времени, например, решением XVI Генеральной ассамблеи Международного астрономического союза, которая проходила с 14 по 25 августа 2006 года в Праге, Плутон больше не является планетой солнечной системы.

Другой пример – оператор *new*, который используется для инстанцирования объектов. Он нарушает LSP, поскольку требует явного указания типа инстацируемго объекта.

В статье [4] описывается ситуация, когда класс Квадрат нельзя наследовать от класса Прямоугольник, так как происходит нарушение LSP.

Данные следствия являются неожиданными и идут вразрез с устоявшимися практиками написания программного кода.

Возникает вопрос: можно ли формально подтвердить или опровергнуть SOLID принципы? Удалось получить математическое обоснование всех 5 принципов с помощью логики Хоара [10].

Алфавитом в логике Хоара является так называемая тройка Хоара

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

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

Были использованы только две аксиомы данной логики:

Аксиома композиции.

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

Аксиома выводимости.

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

Заметим, что поскольку для любого предиката P верно $P \to P$, мы всегда из аксиомы выводимости можем вывести следующие тройки:

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

В отличие от остальных аксиом аксимома выводимости не связана явно ни с какой конструкцией языка программрования, а появилась в логике Хоара, чтоб обеспечить выводимость условной конструкции из обычного выражения.

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 аксимома 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 (5). P. 17–34.
2. *Meyer B.* Object-oriented Software Construction. N.Y.: Prentice Hall, 1988.
3. *Martin R.* The Open-Closed Principle // Object Mentor. 1996. URL: `http://www.objectmentor.com/resources/articles/ocp.pdf`.
4. *Martin R.* The Liskov Substitution Principle // Object Mentor. 1996. URL: `http://www.objectmentor.com/resources/articles/lsp.pdf`.
5. *Martin R.* The Dependency Inversion Principle // Object Mentor. 1996. URL: `http://www.objectmentor.com/resources/articles/dip.pdf`.
6. *Martin R.* The Interface Segregation Principle // Object Mentor. 1996. URL: `http://www.objectmentor.com/resources/articles/isp.pdf`.
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 // Communications of the ACM. 1969. Vol. 12. Is. 10. P. 576–580.