GIẢI PHÁP NÂNG CAO CHẤT LƯỢNG PHẦN MỀM HƯỚNG ĐỐI TƯỢNG

A SOLUTION TO IMPROVE THE QUALITY OF OBJECT-ORIENTED SOFTWARE

NGUYỄN THANH BÌNH-ĐẶNG THỊ LỆ THU

Trường Đại học Bách khoa, Đại học Đà Nẵng

TÓM TẮT

Bài báo trình bày những vấn đề về thiết kế theo hợp đồng, kiểm thử đơn vị cũng như những điểm mạnh và những hạn chế của chúng; từ đó đề ra giải pháp kết hợp thiết kế theo hợp đồng và kiểm thử đơn vị. Công cụ csUnit được sử dụng để kiểm thử đơn vị cho các chương trình viết bằng ngôn ngữ C#. Một thư viện được phát triển để hỗ trợ cho thiết kế theo hợp đồng cho các ngôn ngữ .NET. Bài báo đề xuất giải pháp kết hợp hai phương pháp này nhằm nâng cao chất lượng phần mềm hướng đối tượng.

ABSTRACT

In this paper we present some issues on contract design, unit testing as well as the advantages and disadvantage. Hence, we propose a combination of design by contract and unit testing. The csUnit tool is used to test units implemented in C# language. A library is developed to support the design by contract for .NET languages. The paper shows how the quality of object oriented software is improved when combining the design by contract and unit testing.

1. Đặt vấn đề

Trong lĩnh vực công nghệ phần mềm, phát triển phần mềm ngày càng phức tạp, yêu cầu chất lượng ngày càng cao hơn. Để nâng cao chất lượng phần mềm cần phải cải tiến tất cả các giai đoạn: phân tích, thiết kế, lập trình, kiểm thử, bảo trì. Trong bài báo này, chúng tôi tập trung nghiên cứu các giai đoạn: thiết kế, lập trình và kiểm thử. Một trong những công nghệ hỗ trợ cho giai đoạn thiết kế nhằm đảm bảo tính tin cậy cho phần mềm là thiết kế theo hợp đồng (Design by Contract - DbC). Công nghệ này được dùng cho hệ thống phần mềm phát triển theo hướng đối tượng.

Tuy nhiên, thiết kế theo hợp đồng có những hạn chế nhất định. Những hạn chế này sẽ được trình bày cụ thể sau và sẽ được khắc phục khi dùng kết hợp với kiểm thử đơn vị. Còn hạn chế của kiểm thử đơn vị chính là kiểm thử đơn vị tập trung vào các đơn vị chương trình và không thực hiện trong sự phối hợp giữa các đơn vị đó. Và điều này sẽ được khắc phục khi kết hợp với thiết kế theo hợp đồng. Từ những lý do trên, bài báo này đề xuất giải pháp kết hợp thiết kế theo hợp đồng và kiểm thử đơn vị nhằm nâng cao chất lượng phần mềm hướng đối tượng.

Bài báo được tổ chức như sau. Mục 2 trình bày về thiết kế theo hợp đồng. Kiểm

thử đơn vị được trình bày trong mục 3. Mục 4 nêu các hạn chế của thiết kế theo hợp đồng và kiểm thử đơn vị. Từ đó, đề xuất giải pháp kết hợp thiết kế theo hợp đồng và kiểm thử đơn vị nhằm khắc phục các hạn chế của hai phương pháp trên. Cuối cùng, bài báo kết thúc bởi kết luận.

2. Thiết kế theo hợp đồng

Ngay từ các chương trình máy tính đầu tiên được viết, đã có sự cố gắng để nâng cao tính đáng tin cậy cho phần mềm. Một trong những phương pháp nâng cao tính đáng tin cậy là thiết kế theo hợp đồng do Meyer đề xuất [6], xuất phát cho ngôn ngữ Eiffel. Phương pháp này được dùng cho hệ thống phần mềm phát triển theo phương pháp hướng đối tượng.

Cái nhìn tổng quan đầu tiên về thiết kế theo hợp đồng là cần chú ý rằng tính đúng đắn không phải là thuộc tính của phần mềm, mà là phần mềm có thực hiện đúng hay không so với đặc tả của nó. Các xác nhận (assertion) được dùng để biểu diễn những đặc tả đó.

2.1. Các xác nhận

Một xác nhận là một biểu thức lô-gic bao hàm một số vấn đề của phần mềm và trình bày một thuộc tính mà các vấn đề này phải thỏa mãn khi thực thi phần mềm. Để diễn tả xác nhận, Hoare trình bày công thức của tính đúng đắn [4]: {P} A {Q}

Trong đó, A biểu thị cho một thao tác (operation), {P} là tiền điều kiện (precondition) và {Q} là hậu điều kiện (postcondition). Tiền điều kiện biểu diễn các điều kiện phải đúng bất cứ khi nào A được gọi; hậu điều kiện biểu diễn các điều kiện mà A phải đảm bảo khi thực hiện xong.

2.2. Tiền điều kiện và hậu điều kiện

Tiền điều kiện và hậu điều kiện được sử dụng để định nghĩa ngữ nghĩa các phương thức. Chúng chỉ rõ nhiệm vụ được thi hành bởi một phương thức. Việc định nghĩa tiền điều kiện và hậu điều kiện cho một phương thức là cách để định nghĩa một hợp đồng, hợp đồng này ràng buộc phương thức và các lời gọi đến nó.

Tiền điều kiện mô tả sự ràng buộc mà với sự ràng buộc này, phương thức sẽ thực hiện một cách đúng đắn. Đó là nghĩa vụ của trình khách - trình gọi (client) và là quyền lợi của trình cung cấp (supplier).

Hậu điều kiện diễn tả các thuộc tính từ kết quả thực hiện một phương thức. Đó là nghĩa vụ của trình đáp ứng và là quyền lợi của trình khách.

2.3. Điều kiện bất biến của lớp (class invariant)

Tiền điều kiện và hậu điều kiện biểu diễn các tính chất của mỗi phương thức. Các điều kiện bất biến của lớp mô tả các ràng buộc toàn vẹn của lớp, các ràng buộc này phải được tuân thủ qua tất cả các phương thức trong lớp. Điều kiện bất biến của lớp được thêm vào với tiền điều kiện và hậu điều kiện của mỗi phương thức của lớp:

$$\{INV \ \& \ P\} \ A \ \{INV \ \& \ Q\}$$

INV là điều kiện bất biến được thêm vào. Điều này thể hiện rằng bất biến INV là không thay đổi trước và sau khi thực hiện A.

2.4. Vi phạm hợp đồng

Các hợp đồng mô tả việc quản lý một phương thức với phương thức khác: đây là sự kết nối phần mềm - phần mềm. Chúng biểu diễn các điều kiện đúng đắn. Vì vậy, sự vi phạm hợp đồng là biểu lộ lỗi trong phần mềm: sự vi phạm tiền điều kiện biểu lộ lỗi ở trình khách (trình gọi); sự vi phạm hậu điều kiện biểu lộ lỗi ở trình đáp ứng; sự vi phạm điều kiện bất biến nói lên rằng phương thức đã không được thực hiện một cách phù hợp – điều này biểu lộ lỗi ở trình đáp ứng.

2.5. Lợi ích của hợp đồng

Thiết kế theo hợp đồng yêu cầu viết các xác nhận cùng lúc với việc viết phần mềm. Điều này đem lại những ưu điểm như: tạo được một phần mềm đúng bởi vì nó được thiết kế cho tính đúng đắn; dễ dàng cho việc viết tài liệu phần mềm; cung cấp một nền tảng cho hệ thống kiểm thử và gỡ lỗi.

3. Kiểm thử đơn vị

Phần lớn các phương pháp thiết kế phần mềm đều dẫn đến chia phần mềm thành những mô-đun hay chương trình nhỏ có các dữ liệu vào, kết quả riêng. Chúng ta gọi các mô-đun hay chương trình đó là các đơn vị phần mềm. Đối với phần mềm phát triển theo hướng đối tượng, một đơn vị có thể là một phương thức hay thậm chí một lớp. Kiểm thử riêng rẽ từng đơn vị được gọi là kiểm thử đơn vị. Trong phần lớn các trường hợp, các kỹ thuật kiểm thử chức năng thường được sử dụng để kiểm thử đơn vị. Các dữ liệu thử sẽ thường được tạo ra dựa trên phân tích tài liệu đặc tả, tài liệu thiết kế. Tuy nhiên, đối với các phần mềm đòi hỏi sự chặt chẽ cao, các kỹ thuật kiểm thử tĩnh và kiểm thử cấu trúc được sử dụng.

4. Giải pháp kết hợp thiết kế theo hợp đồng và kiểm thử đơn vị

Những điều kiện của hợp đồng sẽ được kiểm tra và bất cứ sự vi phạm nào trong hợp đồng cũng sẽ được phát hiện. Tuy nhiên, hạn chế của thiết kế theo hợp đồng đó là các xác nhận sẽ trở nên vô ích nếu như phương thức không bao giờ được gọi. Điều này sẽ được khắc phục khi chúng ta dùng kiểm thử đơn vị, bởi kiểm thử đơn vị cho phép kiểm thử khả năng bao phủ mã nguồn. Bên cạnh đó là khả năng tự động và báo cáo tổng kết khi dùng kiểm thử đơn vị. Kết quả của kiểm thử đơn vị được tổng kết và báo cáo một cách dễ dàng.

Có một vài sự khác nhau cơ bản giữa việc kiểm tra xác nhận thiết kế theo hợp đồng và kiểm thử đơn vị. Kiểm thử đơn vị tập trung vào các đơn vị chương trình mà không có sự phối hợp giữa chúng. Đặc biệt, kiểm thử đơn vị là một công cụ nghèo nàn cho việc kiểm tra các tiền điều kiện. Các hạn chế này sẽ được khắc phục khi dùng thiết kế theo hợp đồng.

Từ phân tích trên, nhằm nâng cao chất lượng phần mềm, bài báo này đề xuất

giải pháp sử dụng kết hợp thiết kế theo hợp đồng và kiểm thử đơn vị.

4.1. Giải pháp cho kiểm thử đơn vị

Hiện nay, có các phần mềm được xây dựng sẵn và được tích hợp vào môi trường lập trình để thực hiện việc kiểm thử đơn vị, như JUnit (cho Java), csUnit (cho C#) [2]. Bài báo này sử dụng csUnit để kiểm thử đơn vị cho C#. Trong bài báo này, chúng tôi không giới thiệu một cách chi tiết về csUnit.

CsUnit tránh cho người lập trình phải làm đi làm lại những việc kiểm thử nhàm chán bằng cách tách biệt mã kiểm thử ra khỏi mã chương trình, đồng thời tự động hoá việc tổ chức và thi hành các bộ dữ liệu kiểm thử.

4.2. Giải pháp cho thiết kế theo hợp đồng

Ngôn ngữ .NET nói chung và ngôn ngữ C# nói riêng ngày càng trở nên quan trọng và được sử dụng rộng rãi. Nhưng thật đáng tiếc là các ngôn ngữ phổ biến này lại chưa có các chức năng/công cụ hỗ trợ sẵn có cho thiết kế theo hợp đồng một cách bài bản như Eiffel hay công cụ iContract hỗ trợ cho Java… Vì vậy, ngoài công cụ sẵn có là csUnit hỗ trợ cho kiểm thử đơn vị thì việc xây dựng công cụ hỗ trợ cho thiết kế theo hợp đồng cho .NET là nhu cầu cấp thiết.

Bài báo này này xây dựng công cụ hỗ trợ cho thiết kế theo hợp đồng bằng cách phát triển một thư viện hỗ trợ "hợp đồng". Thư viện này có các chức năng cần thiết hỗ trợ cho các hợp đồng và các quy định cho việc gọi các chức năng này.

Chúng ta có thể xây dựng được thư viện cung cấp một tập các phương thức cho việc định nghĩa các tiền điều kiện, hậu điều kiện, các điều kiện bất biến của lớp. Sau khi xây dựng thư viện, chúng ta có thể tham chiếu đến nó trong trình ứng dụng khách .NET được viết bởi bất kỳ ngôn ngữ nào của .NET. Sau đó, những gì cần làm chính là nạp vào không gian tên DesignByContract. Với mỗi một dự án, khi nạp vào không gian tên DesignByContract, ta có thể:

1. Sinh ra các xác nhận gỡ rối thay cho các ngoại lệ.
2. Cho phép tách biệt hoặc vô hiệu hoá việc kiểm tra các tiền điều kiện, hậu điều kiện, bất biến và các xác nhận.
3. Cung cấp sự mô tả cho mỗi xác nhận, hoặc không. Nếu không cung cấp thì framework sẽ chỉ ra những loại lỗi - tiền điều kiện, hậu điều kiện, … đã sinh ra.
4. Định nghĩa các qui tắc khác cho các bản dịch gỡ rối và các bản dịch phát hành.

Để cho phép hay không cho phép kiểm tra đối với từng loại xác nhận trong hợp đồng, có các đề xuất được đưa ra:

- `DBC_CHECK_ALL` - Kiểm tra các xác nhận – bao gồm việc kiểm tra các tiền điều kiện, hậu điều kiện và các bất biến.
- `DBC_CHECK_INVARIANT` - Kiểm tra các bất biến – bao gồm việc kiểm tra các tiền điều kiện và các hậu điều kiện.
- `DBC_CHECK_POSTCONDITION` - Kiểm tra các hậu điều kiện - bao gồm việc kiểm

tra các tiền điều kiện.

- `DBC_CHECK_PRECONDITION` - Chỉ kiểm tra tiền điều kiện, ví dụ, trong bản dịch phát hành.

Minh hoạ cách dùng thư viện hỗ trợ (viết bằng C#):

```
using DesignByContract;
...
public void Test(int x)
{
  try
  {
    Check.Require(x > 0, "x must be positive.");
  }
  catch (System.Exception ex)
  {
    Console.WriteLine(ex.ToString());
  }
}
```

`Require` là phương thức kiểm tra tiền điều kiện đã được xây dựng trong thư viện, phương thức này thuộc lớp `Check` của không gian tên DesignByContract.

4.3. Ứng dụng giải pháp kết hợp thiết kế theo hợp đồng và kiểm thử đơn vị

Chẳng hạn, cần viết một ứng dụng đơn giản để quản lý dãy các bài báo trong phạm vi một tạp chí. Chúng tôi xây dựng lớp MagazineIndex để có thể làm việc được với một lớp Magazine. Vì vậy, lớp Magazine có thể quản lý một số lớp Article. Công việc trước hết sẽ làm là cần khởi tạo một đối tượng (`cbtMagazineIndex`) của lớp MagazineIndex và thêm một tạp chí (`AddMagazine(…)`) vào đối tượng `cbtMagazineIndex`. Xây dựng mã kiểm thử đơn vị cho hai công việc trên được trình bày như sau:

Mã chương trình để thực hiện các ca kiểm thử đơn vị:

```
using System;
using csUnit;
namespace Magazine
{
    [TestFixture]
    public class TestMagazineIndex
    {
        [Test]
        public void CheckNew()
        {
            MagazineIndex cbtMagazineIndex;
```

```
            cbtMagazineIndex = new MagazineIndex();
            Assert.NotEquals(cbtMagazineIndex, null);
        }
        [Test]
        public void CheckAddMagazine()
        {
            MagazineIndex cbtMagazineIndex;
            cbtMagazineIndex = new MagazineIndex();
            cbtMagazineIndex.AddMagazine("Computerworld Magazine");
             Assert.True(cbtMagazineIndex.IsMember("Computerworld
Magazine"));
        } } }
```

Thuộc tính [TestFixture] được dùng để ra hiệu (cho csUnit) rằng lớp theo sau thuộc tính sẽ bao hàm việc kiểm thử. Trong trường hợp này, lớp TestMagazine được đặt trước với thuộc tính [TestFixture]. Chú ý rằng thuộc tính [TestFixture] chỉ có thể được dùng cho các lớp, không phải cho các phương thức trong phạm vi các lớp. Bên trong lớp TestMagazineIndex, các phương thức được đặt trước với thuộc tính [Test] thì mới được thực hiện kiểm thử. Thuộc tính [Test] chỉ có thể được dùng trong phạm vi các lớp. Các phương thức được đặt trước với thuộc tính [Test] và phải lấy hình thức: public void MyTest( ) thì mới được kiểm thử. Nếu phương thức không thể hiện dưới dạng public void MyTest( ) thì các xác nhận kiểm thử sẽ bị bỏ qua bởi csUnit.

Mục đích kiểm thử mô tả trong đoạn mã chương trình trên là rất rõ ràng:

- Tạo ra một đối tượng của lớp MagazineIndex; kiểm tra rằng đối tượng không trả về giá trị Null.
- Thêm một tạp chí vào lớp MagazineIndex; kiểm tra tạp chí thêm vào là phần tử của lớp MagazineIndex.

Dưới đây là mã chương trình ứng dụng `Magazine`:

```
using System;
using System.Collections;
namespace Magazine
{
    public class MagazineIndex
    {
        public MagazineIndex()
        {
        }
        public void AddMagazine(string MagazineName)
        {
            MagazineList.Add(MagazineName);
        }
```

```
        public bool IsMember(string MagazineName)
        {
            return MagazineList.Contains(MagazineName);
        }
        private ArrayList MagazineList = new ArrayList();
    }
}
```

Thực hiện kiểm thử đơn vị với csUnit. Kết quả cho biết kiểm thử đơn vị không tìm thấy lỗi. Điều đó có đảm bảo rằng chương trình ứng dụng đã hoàn chỉnh ? Để trả lời cho câu hỏi này chúng ta tiếp tục xem lý giải dưới đây.

Rõ ràng là công việc `AddMagazine(string MagazineName)` phải đảm bảo rằng (tiền điều kiện) `MagazineName` không được là chuỗi rỗng. Đồng thời, sau khi thực hiện `AddMagazine(…)` thì phải đảm bảo (hậu điều kiện) số phần tử của `MagazineList` phải tăng lên. Ứng dụng trên đã bỏ qua các ràng buộc này. Lúc này, việc cần làm là thêm vào các xác nhận kiểm tra các ràng buộc trên. Khi đó chúng ta có chương trình với các xác nhận thêm vào theo hợp đồng.

Thiết kế theo hợp đồng cho phương thức `AddMagazine(string MagazineName)`

```
#define DBC_CHECK_PRECONDITION
#define DBC_CHECK_POSTCONDITION
using System;
using System.Collections;
using DesignByContract;
namespace Magazine
{
    public class MagazineIndex
    {
      public void AddMagazine(string MagazineName)
        {
            // Check Pre-Condition
            Check.Require(MagazineName != "", "MagazineName must not
      be empty");
            int oldCount=MagazineList.Count;
            MagazineList.Add(MagazineName);
            // Check Post-Condition
            Check.Ensure(MagazineList.Count > oldCount, "newCount must
      be > oldCount");
        }
     }
 }
```

Khi dùng csUnit để kiểm thử đơn vị, chúng ta nhận được kết quả không tìm thấy

lỗi. Tuy nhiên, kiểm thử đơn vị sẽ hữu ích khi chương trình có lỗi dưới đây. Giả sử phương thức `AddMagazine(…)` xây dựng khác với đặc tả:

```
public void AddMagazine(string MagazineName)
{
      // Check Pre-Condition
      Check.Require(MagazineName != "", "MagazineName must not be empty");
      int oldCount=MagazineList.Count;
      MagazineList.Add(MagazineName.ToUpper());
      // Check Post-Condition
      Check.Ensure(MagazineList.Count > oldCount, "newCount must be > oldCount");
}
```

Đặc tả cho phương thức `AddMagazine(string MagazineName)` chỉ yêu cầu thêm vào một tạp chí mà không yêu cầu phải đổi tên tạp chí `MagazineName` thành in hoa. Thế nhưng người lập trình đã làm một việc *thừa* so với đặc tả là `MagazineList.Add(MagazineName.ToUpper())`. Điều này sẽ gây ra lỗi, lỗi này sẽ bị bỏ qua bởi các xác nhận thiết kế theo hợp đồng. Tuy nhiên khi sử dụng CsUnit sẽ phát hiện ra lỗi này.

CsUnit đã thành công trong việc tìm thấy lỗi tại ví trí CheckAddMagazine. Điều này chứng tỏ phương thức `AddMagazine(string MagazineName)` đã xuất hiện lỗi. Công việc lúc này đơn giản là sửa lại mã chương trình, đổi câu lệnh `MagazineList.Add(MagazineName.ToUpper())` thành `MagazineList.Add(MagazineName.`

Như vậy, ứng dụng trên minh hoạ sự kết hợp thiết kế theo hợp đồng và kiểm thử đơn vị dùng cho ngôn ngữ .NET đã khắc phục được những hạn chế của thiết kế theo hợp đồng cũng như của kiểm thử đơn vị.

5. Kết luận

Giải pháp kết hợp thiết kế theo hợp đồng và kiểm thử đơn vị được áp dụng trên ứng dụng cụ thể cho thấy hiệu quả để phát hiện lỗi. Hai kỹ thuật hỗ trợ cho nhau nhằm phát hiện lỗi một cách tốt nhất. Bài báo đề xuất sử dụng công cụ csUnit kết hợp cùng thư viện hỗ trợ thiết kế theo hợp đồng được xây dựng cho các ngôn ngữ .NET.

Bước đầu, bài báo chỉ mới dừng lại ở ứng dụng minh họa đơn giản, tuy nhiên giải pháp sẽ được tiếp tục triển khai áp dụng trên các ứng dụng phức tạp hơn nhằm đánh giá tốt hơn kết quả thu được. Trong tương lai, việc phát triển một công cụ hỗ trợ hoàn chỉnh kết hợp vừa kiểm thử đơn vị vừa thiết kế theo hợp đồng sẽ được hướng đến.

TÀI LIỆU THAM KHẢO

[1] Rachel Henne-Wu, Dr. William Mitchell, Dr. Cui Zhang. "Support for Design by Contract ™ in the C# Programming Language", *Journal of Object Technology*, vol. 4, no. 7, Septemper – Octorber, 2005.

[2] http://www.csunit.org/

[3] J. E. Payne, J.E., R.T.Alexander and C.D.Hutchinson, *Design for Testability for object oriented Software*, SIGS Publications, 1997.

[4] C.A.R Hoare, *An axiomatic basis for computer programming, Communications of the ACM*, 1969.

[5] Meyer, B., *Applying "design by contract", Computer*, 1992. 25(10): p. 40-51.

[6] Meyer, B., *Design By Contract, in Advances in Object-Oriented Software Engineering*, Prentice Hall, 1991.

[7] Nguyễn Thanh Bình, Phân tích khả năng kiểm thử các đơn vị phần mềm, số 16, *Tạp chí Khoa học và Công nghệ, Đại học Đà Nẵng*, 2006.

[8] Nguyễn Thanh Bình, Huỳnh Phước Danh, Tổng quan về kiểm thử hệ thống hướng đối tượng, *Báo cáo tại Hội thảo Quốc gia lần thứ IX - Một số vấn đề chọn lọc của Công nghệ Thông tin và Truyền thông*, Đà Lạt, tháng 6, 2006.