CODE CONTRACTS VS INPUT VALIDATION

L.Petkova. PhD
University of Library and Information technology (UNIBIT)
lilyanapetkova92@gmail.com

Abstract: *The unstoppable growth of security flaws makes the developers more patient in integrating different security defenses in the application development! Most of the security breaches are due to uninformed or unqualified developers! The good part is that Internet provides a large amount of rules/documentations/guidelines/tools free for use to help the developers in their work! But as the coin has two sides each web application needs to provide two parts of security flaws protection. The first level of defense is the well known protection from the outside world, called user input validation! And the second side is to ensure that the application works without a problem! Which means to protect it from inside out by integrating what is called code contracts! Even thought those two types of protection have similar purposes there is a difference which we are going to present in the research!*
Keywords*: SECURITY, CODE CONTRACTS, CODE RULES, USER INPUT, VALIDATION*

1. Introduction

Working with data should makes us paying attention to the way of managing it! Even though the development life cycle is limited in time and cost that does not mean that, we need to pass through the security of the application from each point of view. By starting from the very first phase of the SDLC until its last stage, we must develop an application, which should resist on the influence of the outside world. [1][2][3]

With this article, we want to introduce general information on two concepts of code validation, which can help in preventing some of the most serious security attacks: code contracts and input validation. Although, their characteristics are common, they have some significant differences, which we present in the current article.

The research is structured in three sections. In the first one, we are analyzing the input validation concept. We provide basic information on its purpose and damages it can cause and some of the most commonly used patterns to guarantee input data validation. The second part of the research contains information on code contracts. Firstly, we are presenting the code contracts idea by providing a classification based on the area of verification. Then we define some advantages of their use and some practices in order to achieve the best of them.

Finally, we are providing a comparison between those two concepts in data validation. As part of our research, so far we are achieving this by proving a code example for each of the provided validation concept.

2. Input Validation

During the evolution of the application's development, we passed through static web site to web sites, which incorporate dynamic data. This data usually flows in two directions: it either comes from the server and is sent to the end user's browser, or the data is entered by the user and sent to the server to be processed or stored.[4][5][10]

Data coming from the server can be retrieved from many different untrusted data sources, including files and databases.

The other flow of data comes from the user and is sent to the server. The underlying principle of this data flow is basically the same in all scenarios — users enter data on a specific topic and then submit it to the server. [4][5][10]

Moreover, the best result in that case is a friendly message returned to the user that something is wrong with the data.

To prevent the application from receiving invalid data, it must be validated before further use! We have already mentioned some of those validation rules, which properly followed, will add the necessary layer of security to the application. [1][3]

Proper input validation can eliminate the vast majority of software vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user-controlled files. [1][3]

There are various services, managers, providers, stores which can help in validating the user input. In general, each application with a client interface and accessed from the outside world must have a proper implementation of validating the user input fields. [9][14][15][16]

The most frequently used validation pattern is the **exception throwing**. It consists of directly checking inputs and throwing exceptions. [14][15][16]

```
void FindUserById(int userId)
{
    if (userId <= 0)  throw new ArgumentException("UserId
must be a greater than 0");
//rest of the body
 }
```

Example 1: *Exception throwing*

Other option is by applying some **regex patterns** to the user input to validate it against some rules. Even though it takes time coming up with a proper regex rule, it is worth the effort! And moreover there are various of rules in the Internet which are ready to be used! [14][16]

```
Regex.Replace(userInput, "[$&+,:;=?@#|'<>.^*()%!]", "")
```

Example 2: *Regex Pattern for remove the special characters from a string*

3. Code Contracts

The code contracts provide a method for expressing constraints and assumptions within the code, which can be validated at both compile time and runtime.

Code contracts have been profoundly used in .NET 4 framework. They have been abandoned as a validation mechanism for the past years. However, in the process of our research we found them really interesting and easy to use and understand. Which makes us want of further investigating them.

3.1. Classification

The code contracts differ based on when they are verified:

Preconditions. Verified when a function starts. [8]

Preconditions specify state when a method is invoked. They are generally used to specify valid parameter values. All members that are mentioned in preconditions must be at least as accessible as the method itself; otherwise, the precondition might not be understood by all callers of a method. The condition must have no side-effects. The run-time behavior of failed preconditions is determined by the runtime analyzer.

```
Contract.Requires<ArgumentNullException>(x != null, "x");
```

Example 3: *Precondition contract[6]*

Postconditions. Verified before a function exits.[8]

The postcondition is checked just before exiting a method. The run-time behavior of failed postconditions is determined by the runtime analyzer.

Postconditions express a condition that must be true upon normal termination of the method.

```
Contract.Ensures(this.F > 0);
```

Example 4: *Postcondition contract[6]*

Unlike preconditions, postconditions may reference members with less visibility. A client may not be able to understand or make use of some of the information expressed by a postcondition using private state, but this does not affect the client's ability to use the method correctly.

Object Variants. Verified after every public function in a class. [8]

Object invariants are conditions that should be true for each instance of a class whenever that object is visible to a client. They express the conditions under which the object is considered correct.

```
[ContractInvariantMethod]
protected void ObjectInvariant ()
{
   Contract.Invariant(this.y >= 0);
   Contract.Invariant(this.x > this.y);
   //rest of the body
}
```

Example 5: *Object Variants[6]*

3.2. Advantages [11][7]

- **Improved testing**: Code contracts provide static contract verification, runtime checking, and documentation generation.
- **Automatic testing tools**: You can use code contracts to generate more meaningful unit tests by filtering out meaningless test arguments that do not satisfy preconditions.
- **Static verification**: The static checker can decide whether there are any contract violations without running the program. It checks for implicit contracts, such as null dereferences and array bounds, and explicit contracts.
- **Reference documentation**: The documentation generator augments existing XML documentation files with contract information. There are also style sheets that can be used with Sandcastle so that the generated documentation pages have contract sections.

3.3. Best practices

Depending on certain rules (preconditions) publicly offered by a service guarantees some results described in the postconditions. Which leads to the following necessary rules of each precondition in order to achieve the best results:

Public. The preconditions should be public so that they can be understandable before event start writing a line of code. [12]

The best way of achieving this is by specifying them at the top of the method's body:

```
public string GetPositionDetails(int position)
{
   Contracts.Require(position, x => x >= 0);
    // Rest of the body
}
```

Example 6: *Publicity*

Simplicity. The preconditions should be easy to check which means that they should not be presented by writing complex algorithms to emulate them. [12]

In case of the need of complex contract, just provide separate methods for each precondition (*one of the best practices at all*).

```
public int Devote(int amount)
{
    Contracts.Require(() => CanDevote(amount));
     return DevoteCore(amount);
}
```

Example 7: *Simplicity*

The precondition contracts should not rely on non-public methods or states because that will restrict client code in its ability to check them.

Stability. The preconditions should be stable. The precondition validation result should not depend on volatile classes or variables.

In the following example, the client depends on the existence of a file, which may be deleted or inaccessible but cannot be checked! That check cannot be defined as a precondition because it cannot be validated.

```
public string ReadFile(string filePath)
{
    Contracts.Require(() => File.Exists(filePath));
  // Rest of the method
}
```

Example 8: *Stability*

In that case we deal with it by adding some exceptions as the one provided in the following code:

```
public string ReadFile(string filePath)
{
    if (!File.Exists(filePath)) throw new ArgumentException();
     // Rest of the body
}
```

Example 9: *Stability*

4. Input validation vs Code Preconditions - results

For the ordinary human being the contract preconditions can be described as a shield placed inside the code to secure the smooth workflow. On the other hand, the validation of the incoming data can be presented as a "defense" against the outside world. [12]

The red point from the Figure 1 represents invalid upcoming request from the outside and the green ones the valid ones.

The main goal of each developer is to take care of that red point to not reach the code!

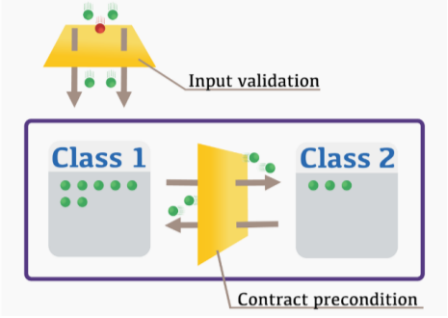


Figure 1: *Input validation vs Contract precondition [12]*

In case the red signal appears inside the application, then it is either not validated input or bug generated in the code. Which from each point of view is must be fixed immediately.

In that case, the code contracts helps in stopping the red signals from spreading across the application shown on Figure 2.

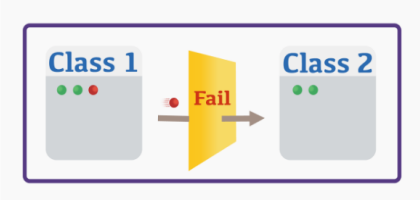


Figure 2: *Precondition violation [12]*

The input validation protects the application from invalid input data, but not on the data coming in the application. Which makes it a valid situation with invalid data. [12]

On the contrary, the contract preconditions validate the data from inside the application, which makes it a bug in the system in case of data violation. [12]

For better understanding the difference between the input validation and the code contract, we are going to review an example of a method returning the difference between two integers.

The rules are as follow: [13]

- The minimal value should be greater or equal to 0
- The maximum value should be greater or equal to 0
- The minValue should be smaller than the maxValue
- The result should be smaller than the maximum value

With the following code from Example 10, we are providing the code concept of input validation:

```
public int GetNumbersDifference(int minValue, int maxValue)
 {
    if (minValue >= 0)
    {
       throw new ArgumentException("minValue");
    }
    if (maxValue >= 0)
    {
      throw new ArgumentException("minValue");
    }
   if (!(minValue < maxValue))
    {
      throw new ArgumentException("minValue, maxValue");
    }
  var result = maxValue - minValue;
   if (!(result <= maxValue))
  {
     throw new Exception("Something went wrong");
  }
 return result;
}
```

Example 10: *Input Validation Example[13]*

With code contracts, the code from Example 10 would be converted to something like the code from Example 11:

```
public int GetNumbersDifference(int minValue, int maxValue)
 {
Contract.Requires<ArgumentException>(minValue >= 0);
Contract.Requires<ArgumentException>(maxValue >= 0);
Contract.Requires<ArgumentException>(minValue<maxValue);
Contract.Ensures(Contract.Result<int>() <= maxValue);
return maxValue - minValue;
 }
```

Example 11: *Code Contract Example[13]*

At first sight, the second code provide much more readable and easy for the ordinary human to understand concept!

5. Conclusion

The need of code validation is the most significant step in the developer's work. The two concepts presented in this research although common has their own characteristics. Moreover, although we have given the option to choose which one to use, one of those possibilities is now abandoned which is not recommended to use in production code bases! That of course does not allow the developer to abandon the protection of the application. As the code validation is the most protective step in the security integration of the application! Moreover, its need will always be in favor.

References

[1] Petkova, L., SECURITY STANDARDS in software development, 2017
[2] Petkova, L., HTTP SECURITY HEADERS, 2018
[3] Petkova, L., CONTENT SECURITY POLICY VALIDATION, 2019
[4] Petkova, L., SEO SPAM AND MALWARE, 2019
[5] Petkova, L., SECURITY'S LEAKS IN SEO SPAMMING, 2019
[6] Mackey, Alex, Introducing .NET 4.0: With Visual Studio 2010
[7] Kolev, Rosen, Validation Rule Patterns in C#, FEBRUARY 2018
[8] Roque Alex, Web Application Programming, Lectures, 2016
[9] Code Contracts, Microsoft, May 2018
[10] Code Contracts User Manual, Microsoft Manual, January, 2012
[11] Albahari, Joseph, Code Contracts, 2007-2017
[12] Khorikov, Vladimir, C# code contracts vs input validation, February 2015
[13] Hakan Onur, Code Contracts – A brilliant way to do validation in your code and even do more, November 2010
[14] ICT Security Trends, Willian Dimitrov, Sofia, 2017, Avangard, ISBN 978-619-160-766-2
[15] Software testing, Willian Dimitrov, Sofia, 2017, Avangard, ISBN 978-619-160-765-5
[16] ICT Security Model, Willian Dimitrov, Sofia, 2018, Avangard, ISBN 978-619-160-950-5