

# Code Review

Юрий Литвинов

yurii.litvinov@gmail.com

10.04.2018г

Code review (инспекция кода, ревизия кода) — это систематическая проверка исходного кода с целью обнаружения и исправления ошибок, направленная как на повышение качества ПО, так и навыков разработчика. В процессе ревью находятся не только и не столько ошибки, связанные с функционированием программы, сколько ошибки архитектуры, оформления, уязвимости к хакерству, утечки памяти, просто некрасивый код. В процессе ревью может быть выявлено более половины труднообнаружимых ошибок. Как проводится: бывают формальные процессы ревью кода (например, в некоторых крупных компаниях и во многих проектах с открытым исходным кодом код не может быть принят в основную ветку без ревью), либо же неформальные, когда код для ревью рассылается всем участникам, после чего они собираются вместе и разбирают замеченные ошибки. Некоторые системы управления проектами, такие как Github, позволяют писать комментарии к коммитам и пуллреквестам, “защищать” ветки, требуя хотя бы одного положительного отзыва перед тем, как пуллреквест в принципе можно будет замержить.

Правила проведения кодревью такие:

- Код безличен, то есть ревьюится код, а не его автор. Автор может быть (и, как правило, обязательно будет) в курсе обсуждения, поэтому высказывать сомнения в его профессионализме или умственных способностях запрещается. Помните, что вы можете оказаться на его месте, и не факт, что вы пишете лучше.
- В общепринятых сейчас методологиях разработки пропагандируется коллективное владение кодом, при котором всем кодом, лежащим в основной ветке, владеет вся команда, несёт за него равную ответственность и имеет равные права на его исправление. Это несколько помогает процессу ревью и последующего рефакторинга.
- Кодревью — это не деструктивная, а конструктивная деятельность. Мы не указываем на недостатки/ошибки и т.д., а обязательно предлагаем пути по исправлению этих ошибок.

Код для ревью, пример 1:

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;
```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Calculator
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            /// <summary>
            /// Procedure of adding symbol in textbox
            /// </summary>
            /// <param name="value">Value to add</param>
            public void AddSymbol(int value)
            {
                if (AlreadyPrinted)
                {
                    textBox1.ResetText();
                    AlreadyPrinted = false;
                }
                textBox1.Text = textBox1.Text + value;
                label2.Text = "";
            }

            /// <summary>
            /// Symbol of the operation
            /// </summary>
            private char OperationSymbol = 'x';

            /// <summary>
            /// If symbol is printed
            /// </summary>
            private bool AlreadyPrinted = false;

            /// <summary>
            /// Previous value
            /// </summary>
            private int Previous = 0;

            //Buttons with numbers clicked methods
            private void OnButton1Click(object sender, EventArgs e)

```

```
{  
    this.AddSymbol(1);  
}  
  
private void OnButton2Click(object sender, EventArgs e)  
{  
    this.AddSymbol(2);  
}  
  
private void OnButton3Click(object sender, EventArgs e)  
{  
    this.AddSymbol(3);  
}  
  
private void OnButton4Click(object sender, EventArgs e)  
{  
    this.AddSymbol(4);  
}  
  
private void OnButton5Click(object sender, EventArgs e)  
{  
    this.AddSymbol(5);  
}  
  
private void OnButton6Click(object sender, EventArgs e)  
{  
    this.AddSymbol(6);  
}  
  
private void OnButton7Click(object sender, EventArgs e)  
{  
    this.AddSymbol(7);  
}  
  
private void OnButton8Click(object sender, EventArgs e)  
{  
    this.AddSymbol(8);  
}  
  
private void OnButton9Click(object sender, EventArgs e)  
{  
    this.AddSymbol(9);  
}  
  
private void OnButton0Click(object sender, EventArgs e)  
{
```

```

        this.AddSymbol(0);
    }
    /// <summary>
    /// Save previous value
    /// </summary>
    public void SavePrev()
    {
        if (textBox1.Text == "")
            return;
        Previous = int.Parse(textBox1.Text);
        AlreadyPrinted = true;
        label2.Text = "";
    }

    ///Buttons with symbols clicked methods
    private void OnButtonMinusClick(object sender, EventArgs e)
    {
        SavePrev();
        OperationSymbol = '-';
    }

    private void OnButtonPlusClick(object sender, EventArgs e)
    {
        SavePrev();
        OperationSymbol = '+';
    }

    private void OnButtonMultiplyClick(object sender, EventArgs e)
    {
        SavePrev();
        OperationSymbol = '*';
    }

    private void OnButtonDivideClick(object sender, EventArgs e)
    {
        SavePrev();
        OperationSymbol = '/';
    }

    private void buttonEquals_Click(object sender, EventArgs e)
    {
        int result = 0;
        if ((OperationSymbol == 'x') || (textBox1.Text == ""))
            return;
        if (OperationSymbol == '+')
        {

```

```

        result = Previous + int.Parse(textBox1.Text);
        textBox1.ResetText();
        textBox1.Text = textBox1.Text + result;
    }
    if (OperationSymbol == '-')
    {
        result = Previous - int.Parse(textBox1.Text);
        textBox1.ResetText();
        textBox1.Text = textBox1.Text + result;
    }
    if (OperationSymbol == '*')
    {
        result = Previous * int.Parse(textBox1.Text);
        textBox1.ResetText();
        textBox1.Text = textBox1.Text + result;
    }
    if (OperationSymbol == '/')
    {
        if (int.Parse(textBox1.Text) == 0)
        {
            label2.Text = "ERROR";
            textBox1.ResetText();
            Previous = 0;
            OperationSymbol = 'x';
            AlreadyPrinted = false;
            return;
        }
        result = Previous / int.Parse(textBox1.Text);
        textBox1.ResetText();
        textBox1.Text = textBox1.Text + result;
    }
    Previous = int.Parse(textBox1.Text);
    OperationSymbol = 'x';
    AlreadyPrinted = false;
}

/// <summary>
/// Button C click method
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void OnButtonCClick(object sender, EventArgs e)
{
    textBox1.ResetText();
    Previous = 0;
}
}

```

```
}
```

Код для ревью, пример 2:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Queue_
{
    class Program
    {
        static void Main(string[] args)
        {
            Queue queue = new Queue();

            queue.Enqueue(2, 1);
            queue.Enqueue(8, 1);
            queue.Enqueue(5, 3);
            int tempVal = queue.Dequeue();
            tempVal = queue.Dequeue();
            tempVal = queue.Dequeue();
        }
    }

    public class Queue
    {
        public Qelement qBeg = null;
        public Qelement qEnd = null;
        public class Qelement
        {
            private int val;
            public int Value
            {
                get
                {
                    return val;
                }
                set
                {
                    val = value;
                }
            }
            public int priority;
        }
    }
}
```

```

    public Qelement Next { get; set; }
}

/// <summary>
/// Проверяет очередь на пустоту.
/// </summary>
/// <returns></returns>
public bool IsEmpty()
{
    return qBeg == null;
}

/// <summary>
/// Находит позицию для добавления нового элемента в очередь с приоритетами.
/// </summary>
/// <param name="priority"></param>
/// <returns></returns>
private Qelement SearchPos(int priority)
{
    Qelement pos = qBeg;
    while (pos != null && priority < pos.priority - 1)
    {
        pos = pos.Next;
    }
    return pos;
}

/// <summary>
/// Добавляет новый элемент в очередь с приоритетами.
/// </summary>
/// <param name="val"></param>
/// <param name="priority"></param>
public void Enqueue(int val, int priority)
{
    Qelement pos = SearchPos(priority);
    if (pos != null)
    {
        Qelement newEl = new Qelement()
        {
            Next = pos.Next,
            Value = val
        };
        pos.Next = newEl;
        if (pos == qBeg && priority > qBeg.priority)
        {
            int temp = pos.Value;

```

```

        pos.Value = newEl.Value;
        newEl.Value = temp;
    }
}
else
{
    Qelement newEl = new Qelement()
    {
        Next = null,
        Value = val
    };
    if (!IsEmpty())
    {
        qEnd.Next = newEl;
        qEnd = newEl;
    }
    else
    {
        qBeg = newEl;
        qEnd = newEl;
    }
    newEl.priority = priotity;
}

}

/// <summary>
/// Возвращает значение с наивысшим приоритетом и удаляет его из очереди.
/// </summary>
/// <returns></returns>
public int Dequeue()
{
    if (!IsEmpty())
    {
        int temp = qBeg.Value;
        qBeg = qBeg.Next;
        return temp;
    }
    throw new EmptyQueueExeption();
}

}

/// <summary>
/// Исключение возникает при попытке удалить элемент из пустой очереди.
/// </summary>

```



```

public class EmptyQueueException : ApplicationException
{
    public EmptyQueueException()
    {
    }
    public EmptyQueueException(string message)
        : base(message)
    {
    }
}
}

```

Код для ревью, пример 3:

```

template <typename T>
class BST
{
public:
    struct Node
    {
        T item;
        Node* left;
        Node* right;
        Node(T value) : left(nullptr), right(nullptr) {item = value;}
        Node() : left(nullptr), right(nullptr) {}
        ~Node()
        {
            if (!left)
                delete left;
            if (!right)
                delete right;
        }
    };

    typedef Node* Position;

    BST()
    {
        root = new Node();
        counter = 0;
    }

    ~BST()
    {
        //call destructor of Node structure
        delete root;
    }
}

```

```

}

//add element to the tree
void insert(T value)
{
    inject(root, value);
    ++counter;
}

//get pointer to element of tree with needed key
Position find(T value)
{
    Position search = root;
    while (search != nullptr && search->item != value)
    {
        if (search->item <= value)
            search = search->right;
        else
            search = search->left;
    }
    return search;
}

//delete item from the tree
void remove(T toRemove)
{
    if (counter != 0)
    {
        detach(toRemove, root);
    }
}

void traverseUp(void (*action)(T))
{
    auxTraverseUp(root, action);
}

void traverseDown(void (*action)(T))
{
    auxTraverseDown(root, action);
}

private:
    Node* root;
    size_t counter;

```

*//Recursively add value to tree*

```
void inject(Node* toAdd, T value)
{
    if (counter == 0)
    {
        //empty-tree case
        root = new Node(value);
        return;
    }
    if (toAdd->item <= value)
        if (toAdd->right != nullptr)
            inject(toAdd->right, value);
        else
            toAdd->right = new Node(value);
    else
        if (toAdd->left != nullptr)
            inject(toAdd->left, value);
        else
            toAdd->left = new Node(value);
}
```

*//Recursively delete value from tree*

```
void detach(T value, Node* tree)
{
    if (root->item == value)
    {
        removeNode(root);
        --counter;
        return;
    }
    if (tree->left != nullptr && tree->left->item == value)
    {
        tree->left = removeNode(tree->left);
        --counter;
        return;
    }
    if (tree->right != nullptr && tree->right->item == value)
    {
        tree->right = removeNode(tree->right);
        --counter;
        return;
    }
    if (tree->item > value)
    {
        if (tree->left != nullptr)
            detach(value, tree->left);
    }
}
```

```

    }
    else
    {
        if (tree->right != nullptr)
            detach(value, tree->right);
    }
    return;
}

//exclude Node from tree
Node* removeNode(Node* toDelete)
{
    short hasRight = 5 * static_cast<short>(toDelete->right != nullptr);
    short hasLeft = 3 * static_cast<short>(toDelete->left != nullptr);
    Position temp;
    /*depends on values of hasRight and hasLeft,
    their sum can be either 0 or 3 or 5 or 8, as far as
    static_cast<short>(toDelete->right != nullptr) can return either 0 or 1*/
    switch (hasLeft + hasRight)
    {
        case 0:
            //has no childs
            delete toDelete;
            return nullptr;
            break;
        case 3:
            //has left child
            temp = toDelete->left;
            delete toDelete;
            return temp;
            break;
        case 5:
            //has right child
            temp = toDelete->right;
            delete toDelete;
            return temp;
            break;
        case 8:
            //has both childs
            temp = toDelete->right;
            while (temp->left != nullptr)
                temp = temp->left;
            toDelete->item = temp->item;
            toDelete->right = removeNode(temp);
            return toDelete;
            break;
    }
}

```

```

    }
    return 0;
}

//traverse to max element with some function
void auxTraverseUp(Node* node, void (*action)(T))
{
    if (node->left != nullptr)
        auxTraverseUp(node->left, action);
    action(node->item);
    if (node->right != nullptr)
        auxTraverseUp(node->right, action);
    return;
}

//traverse to max element with some function
void auxTraverseDown(Node* node, void (*action)(T))
{
    if (node->right != nullptr)
        auxTraverseDown(node->right, action);
    action(node->item);
    if (node->left != nullptr)
        auxTraverseDown(node->left, action);
    return;
}
};

```

Код для ревью, пример 4. Файл tree.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ParseTree
{
    public class Tree
    {
        private string[] tokens;
        private Dictionary<string, Func<double, double, double>> operation;
        private int counter;

        /// <summary>
        /// Initializes a new instance of the class
        /// </summary>
        /// <param name="path"></param>
    }
}

```

```

public Tree(string path)
{
    string expression = "";
    using (StreamReader f = new StreamReader(path))
    {
        expression = f.ReadLine();
    }
    this.operation = new Dictionary<string, Func<double, double, double>>
    {
        { "+", (x, y) => x + y },
        { "-", (x, y) => x - y },
        { "*", (x, y) => x * y },
        { "/", (x, y) => x / y }
    };
    this.tokens = expression.Split(new char[] { ' ' });
    this.counter = 0;
}

public double Calculate(Node tree)
{
    return tree.Calculate();
}

public void PrintTree(Node tree)
{
    tree.Print();
}

/// <summary>
/// Build tree
/// </summary>
public void Build(ref Node current)
{
    string token = tokens[counter++];
    if (token == ")")
    {
        return;
    }

    if (operation.ContainsKey(token))
    {
        if (current != null)
        {
            var currentOperation = current as NodeOperation;
            Node newCurrent = currentOperation.AddOperand(
                new NodeOperation(token, operation[token]));
        }
    }
}

```

```

        Build(ref newCurrent);
    }
    else
    {
        current = new NodeOperation(token, operation[token]);
    }
}
else
{
    double value;
    if (double.TryParse(token, out value))
    {
        if (current != null)
        {
            (current as NodeOperation).AddOperand(new NodeOperand(value));
        }
        else
        {
            current = new NodeOperand(value);
        }
    }
}
Build(ref current);
}
}
}
}

```

Файл node.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ParseTree
{
    public interface Node
    {
        double Calculate();

        void Print();
    }
}

```

Файл nodeOperand.cs:

```

using System;
using System.Collections.Generic;

```

```

using System.Linq;
using System.Text;

namespace ParseTree
{
    public class NodeOperand : Node
    {
        private double value;

        /// <summary>
        /// Initializes a new instance of the class.
        /// </summary>
        /// <param name="value"></param>
        public NodeOperand(double value)
        {
            this.value = value;
        }

        public void Print()
        {
            Console.WriteLine(string.Format(" {0} ", this.value));
        }

        public double Calculate()
        {
            return value;
        }
    }
}

```

Файл nodeOperation.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ParseTree
{
    public class NodeOperation : Node
    {
        private Node left;
        private Node right;
        private string signature;
        private Func<double, double, double> perform;

        /// <summary>

```



```

    /// Initializes a new instance of the class.
    /// </summary>
    public NodeOperation(string sign, Func<double, double, double> performOperation)
    {
        this.perform = performOperation;
        this.signature = sign;
    }

    public double Calculate()
    {
        return perform(left.Calculate(), right.Calculate());
    }

    public void Print()
    {
        Console.Write("{0} ", signature);
        left.Print();
        right.Print();
        Console.Write(" ");
    }

    /// <summary>
    /// Add an operand
    /// </summary>
    public Node AddOperand(Node operand)
    {
        if (left == null)
        {
            return left = operand;
        }
        else
        {
            if (right == null)
            {
                return right = operand;
            }
            else
            {
                throw new ExtraNodeException();
            }
        }
    }
}
}

```