
# WORK LOG 9

**Shandong University**

**May 3, 2020**

高德琛

# Contents

# 1 故障分析

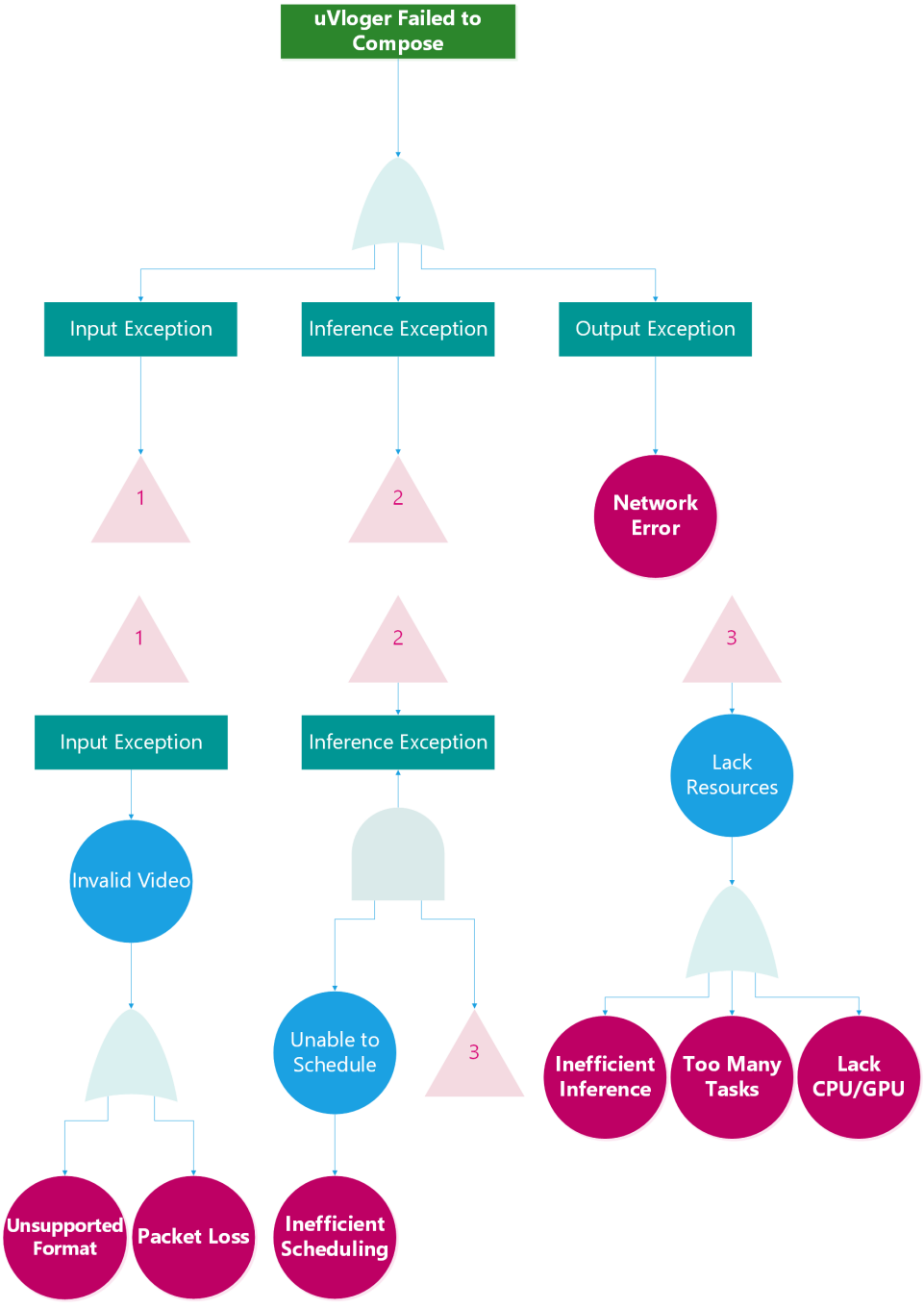


图 **1:** FTA

根据图 1求割集，首先给事件以编号。

| | |
|---|---|
| Unsupported Format | $x_1$ |
| Packet Loss | $x_2$ |
| Inefficient Scheduling | $x_3$ |
| Inefficient Reference | $x_4$ |
| Too Many Tasks | $x_5$ |
| Lack GPU/CPU | $x_6$ |
| Network Error | $x_7$ |
| Input Exception | $a_1$ |
| Inference Exception | $a_2$ |
| Output Exception | $a_3$ |

表 **1:** 事件表

根据表 1进行运算，得到割集划分。

$$\begin{aligned} T &= a_1 + a_2 + a_3 \\ &= (x_1 + x_2) + (x_3(x_4 + x_5 + x_6)) + x_7 \\ &= x_1 + x_2 + x_3x_4 + x_3x_5 + x_3x_6 + x_7 \end{aligned}$$

# 2 软件架构笔记

## 2.1 基础定义

- 对于软件架构定义有很多种，通用的定义是：某个软件或计算机系统的软件架构是该系统的一个或多个结构，他们由软件元素，这些元素的外部可见属性以及这些元素之间的关系组成。

  这里所说的某个元素的“外部可见属性”是指其他元素对该元素所做的假设，如它所提供的服务、性能特征、错误处理、共享资源的使用，等等。

- 其他的定义包括：架构是一种高层设计。架构是系统的总体结构。架构是一个软件或系统的组件、组件之间的相互关系以及管理其设计和演变的原理和方针的结构。架构是组件和连接器。

- 架构模式是对元素和关系类型以及一组对其使用方式的限制的描述。

- 参考模型是一种考虑数据流的功能划分。

- 参考架构是映射到软件元素（它们相互协作，共同实现在参考模型中定义的功能）及元素之间数据流上的参考模型。

## 2.2 架构意义

- 架构是涉众进行交流的手段。
- 架构是可传递、可重用的模型。
- 架构是早期设计决策的体现。

## 2.3 质量属性

系统从设计、实现到部署的整个过程中考虑质量属性的实现。质量属性包括下列三类:

- 系统的质量属性。(可用性、可修改性、性能、安全性、可测试性和易用性)
- 受架构影响的商业属性。(上市时间、成本和收益、所希望的系统生命期的长短、目标市场、推出计划、与老系统的集成)
- 与架构本身相关的一些质量属性。(概念完整性、正确性与完整性、可构建性)

## 2.4 战术与架构模式

Active Objcet 设计模式将方法执行从方法调用中分离出来，以增强并发，并简化对驻留在其自身控制线程中的对象的同步访问。

该模式由 6 个元素组成：代理，它提供了允许客户对主动对象调用公共访问方法的接口；方法请求，它定义了用于执行主动对象的方法的一个接口；激活接口，它维持了挂起方法请求的一个缓冲器；调度程序，它决定接下来执行什么方法请求；附属，他定义可建模为主动对象的行为和状态；将来，它允许客户获得方法调用的结果。

该模式的动机就是增强并发性——这是一个性能目标。因此其主要目的就是实现“引入并发“性能战术。然而，还要注意该模式包含的其他战术。

- 信息隐藏（可修改性)。每个元素都选择了它将实现的责任，并将其实现隐藏在接口后面。
- 仲裁者（可修改性)。该代理充当着把变化缓冲到方法调用中的仲裁者。

- 绑定时间（可修改性）。主动对象模式假定对该对象的请求在运行时到达该对象。然而，并没有确定客户机与代理的绑定时间。
- 调度策略（性能）。调度程序实现一些调度策略。
- 对设计师来说，分析过程包括理解嵌入在实现中的所有战术；设计过程包括在关于哪些战术最和将实现系统期望的目标方面，做出一个明智的选择。

## 2.5 架构模式和样式

软件中架构模式与建筑物中的架构样式类似，它由几个将他们组合起来以维持架构完整性的关键特性和规则组成。架构模式由以下几个因素确定：

- 一组元素类型（如数据存储库或计算数学函数的组件）。
- 指出其相互关系的元素的拓扑布局。
- 一组语义限制（如管道——过滤器样式中的过滤器是纯数据转化器——他们以增量形式将其输入流转换为输出流，但并不控制上游流或下游元素）。
- 一组交互机制（如子例程调用、事件——调阅者、黑板）、他们确定元素将如何通过允许的拓扑进行协调。

架构模式和战术之间是什么关系呢？正如已经说明的那样，我们把战术看作是设计的基本“构建块”，并根据该战术创建架构模式和策略。

## 2.6 设计架构

几乎在我们遇到的所有成功的面向对象系统中都具有但失败的系统中缺少的两个特性是：存在一个强大的构架构想，应用管理良好的迭代式增量开发周期。

功能、质量和商业需求的某个集合“塑造”了构架。我们把这些塑造需求称为“构架驱动因素”

。

构架设计必须按需求分析进行，但不需要在需求分析完成后在开始构架设计。实际上，在确定了关键的构架驱动因素后，就可以开始构架设计了。当设计了构架的足够多的部分后，就可以开始开发骨架系统了。该骨架系统在上面进行迭代开发（以及其在任何一个点交付的能力）的框架。组织结构对构架产生影响。

属性驱动的设计 (ADD) 是一个用于设计构架以满足质量需求和功能需求的方法。ADD 把一组质量属性场景作为输入，并使用对质量属性实现和构架之间的关系的了解，对构架进行设计。

ADD 步骤：

- 选择要分解的模块。
- 根据这些步骤对模块进行求精。
- 对需要进一步分解的每个模块重复上述步骤。

## 2.7 构架编档

构架编档是创建构架的最有价值的一步。即使构架非常完美，但如果没有人理解它，或主要的涉众误解了它，它也没有什么用处。如果您创建了一个非常强大的构架，那么，就必须用足够的细节明确地描述它，并以一种其他人可以快速找到所需要信息的方式对其进行组织。否则，构架不会有什么用处，您所付出努力就白费了。捕获构架信息的描述方式采用 UML 表示法。

## 2.8 架构重构

构架重构是一种解释、交互和迭代的过程，涉及许多活动；它不是自动进行的。它需要反向工程专家和设计师具备相关技能并投入精力，这在很大程度上是因为没有在源代码中清楚地表示构架构件。

软件构架重构由以下活动组成，这些活动以迭代的方式进行：

- 信息提取。
- 数据库构造。
- 视图融合。
- 重构。

## 2.9 分析架构

构架评估的一些基本问题——原因、时间、成本、收益、技巧、计划内、计划外、前置条件以及结构。

每个基于构架的开发方法中都应该进行构架评估。
在生命周期中尽可能早的评估软件质量几乎总是经济高效的。
评估成本就是需要参与评估的人员所付出的时间。
成功评估应该具有如下属性:

- 表述清楚的构架目标和需求。
- 可控制的范围。
- 经济高效。
- 关键人员的可用性。
- 称职的评估小组。
- 可管理的期望。

# 3 主程序/子程序 KWIC 系统

## 3.1 定义

本部分给出主程序/子程序风格（Main Program/Subroutine Style）KWIC 实现。

主程序/子程序风格将系统组织成层次结构，包括主程序、一些列子程序。主程序作为 controller，调度各个子程序，子程序又作为其局部的 controller。

## 3.2 分析

主程序/子程序工作流风格得优点在于：与实际业务处理系统的结构契合，逻辑上结构易于理解，支持变换服用。既可以理解成现行的系统，也可以实现为并发系统。

缺点在于通信变换得数据必须有所协议，每个变换必须解析其输入并处理为协议规定得格式进行输出。而这一部分是无法避免的系统负荷，意味着不可能复用使用不兼容数据结构的函数变换。

## 3.3 实现

```
package cz.cvut.fel.ass.kwic.sharedData;

import java.io.*;
import java.util.Arrays;

public class KWIC {

    /**
     * Keeps all the characters.
     */
    private char[] chars = new char[1000];
    private int charCounter = 0;

    /**
     * Keeps indexes of each line starting character.
     */
    private int[] lineIndexes = new int[10];

    /**
     * Keeps 2D array of indexes of circular shifts of the lines.
     * First key is start of the line from lineIndexes.
     * Second index marks where the line starts.
     */
    private int[][] wordIndexes;
    private int wordCounter = 0;

    /**
```

```
     * Keeps 2D array of indexes of alphabetically sorted
     * circular shifts of the lines.
     * First key is start of the line from lineIndexes.
     * Second index marks where the line starts.
     */
    private int[][] alphabetizedIndexes;

    /**
     * KWIC constructor.
     */
    public KWIC() {
        Arrays.fill(lineIndexes, -1);   // Initialize all line indexes to -1
        Arrays.fill(chars, (char) -1);  // Initialize all chars to -1
    }

    /**
     * Read input characters from file and saves it to shared storage.
     * Input Component.
     * <p>
     * Reads input file and saves its characters to shared "chars" array.
     * It also marks starts of each line by
     * saving indexes of "chars" on which the line starts
     * to "lineIndexes" array.
     * </p>
     *
     * @throws IOException
     */
    public void input(Reader reader) throws IOException {
        int inputChar;
```

```
        int lineIterator = 0;
        boolean newLine = true;

        while ((inputChar = reader.read()) != -1) {
            // Mark start of the line
            if (newLine) {
                lineIndexes = expandIfNeeded(lineIndexes,
                lineIterator + 1);
                lineIndexes[lineIterator++] = charCounter;
                newLine = false;
            }

            // If this is the end of line, mark it and rewrite
            // the character to space
            // (this avoids problems while outputting the lines)
            if (inputChar == '\n') {
                newLine = true;
                inputChar = ' ';
            }

            // Save the character
            chars = expandIfNeeded(chars, charCounter + 1);
            chars[charCounter++] = (char) inputChar;
        }
    }

    /**
     * Circular shifts the lines.
     * Circular Shift Component.
```

```
     * <p/>
     * Iterates through lines and searches for words.
     * Each start of the word is then indexed to wordIndexes.
     * Each circular shift begins with different word on the line,
     * therefore each word index is also an index
     * of circular shift.
     * <p/>
     */
    public void circularShift() {
        // Allocate shift indexes
        wordIndexes = new int[lineIndexes.length][30];

        // Iterate through lines
        int lineIterator = 0;
        while (lineIterator + 1 < lineIndexes.length &&
                lineIndexes[lineIterator] != -1) {
            int lineStart = lineIndexes[lineIterator];
            int lineEnd = lineIndexes[lineIterator + 1] != -1 ?
                        lineIndexes[lineIterator + 1] : charCounter;

            Arrays.fill(wordIndexes[lineIterator], -1);

            // Iterate through the line chars and look for new words
            int indexIterator = 0;
            boolean newWord = true;
            for (int charIterator = lineStart; charIterator < lineEnd;
                 ++charIterator) {
                // If this is the end of word, skip it
                if (chars[charIterator] == ' ') {
```

```
                newWord = true;
                continue;
            }

            // Mark start of new word
            if (newWord) {
                wordIndexes[lineIterator] = expandIfNeeded(
                wordIndexes[lineIterator],
                indexIterator + 1);
                wordIndexes[lineIterator][indexIterator++] =
                charIterator;
                ++wordCounter;
                newWord = false;
            }
        }

        // Bump line iterator
        ++lineIterator;
    }
}

/**
 * Alphabetically orders all circular shifts of
 * each line in the file.
 * Alphabetizer Component.
 */
public void alphabetizer() {
    // Fill up alphabetized indexes
    alphabetizedIndexes = new int[wordCounter][3];
```

```
int index = 0;
for (int line = 0; line < lineIndexes.length &&
     lineIndexes[line] != -1; ++line) {
    for (int word = 0; word < wordIndexes[line].length &&
         wordIndexes[line][word] != -1; ++word, ++index) {
         // Line index
        alphabetizedIndexes[index][0] = line;
        // Word index in line
        alphabetizedIndexes[index][1] = word;
        // Word index in chars
        alphabetizedIndexes[index][2] = wordIndexes[line][word];
    }
}

// Iterate through words
for (int bubbleIndex = 0; bubbleIndex < alphabetizedIndexes.length
     - 1; ++bubbleIndex)
    for (index = 0; index < alphabetizedIndexes.length -
         bubbleIndex - 1; ++index) {
        int wordIndex = wordIndexes[alphabetizedIndexes[index][0]]
        [alphabetizedIndexes[index][1]];
        int nextWordIndex = wordIndexes
        [alphabetizedIndexes[index + 1][0]]
        [alphabetizedIndexes[index + 1][1]];

        char wordChar = (char) -1;
        char nextWordChar = (char) -1;

        // Read chars until they aren't equal
```

```
                for (; wordChar == nextWordChar &&
                     nextWordIndex < chars.length &&
                     chars[nextWordIndex] != -1;
                     ++wordIndex, ++nextWordIndex) {
                    wordChar = chars[wordIndex];
                    nextWordChar = chars[nextWordIndex];
                    ++wordIndex;
                    ++nextWordIndex;
                }

                // Now that chars aren't equal, compare them,
                // then do the bubble switch
                if (wordChar > nextWordChar) {
                    // Bubble sort switch
                    int[] temp = alphabetizedIndexes[index];
                    alphabetizedIndexes[index] =
                    alphabetizedIndexes[index + 1];
                    alphabetizedIndexes[index + 1] = temp;
                }
            }
    }

    /**
     * Outputs alphabetically sorted circular shifts of the lines.
     * Output Component.
     */
    public void output(Writer writer) throws IOException {
        for (int index = 0; index < alphabetizedIndexes.length; ++index)
        {
```

```
            int wordStart = alphabetizedIndexes[index][2];
            int lineStart = lineIndexes[alphabetizedIndexes[index][0]];
            int lineEnd = lineIndexes[alphabetizedIndexes[index][0] + 1]
            != -1 ? lineIndexes[alphabetizedIndexes[index][0] + 1] :
            charCounter;

            // From the word start to the end of line
            for (int charIndex = wordStart; charIndex < lineEnd;
                 ++charIndex) {
                writer.write(chars[charIndex]);
            }
            // From the beginning of the line to the word
            for (int charIndex = lineStart; charIndex < wordStart;
                 ++charIndex) {
                writer.write(chars[charIndex]);
            }
            writer.write('\n');
        }
        writer.flush();
    }

    /**
     * Expands the given array if the given index is out of bounds.
     *
     * @param array original array
     * @param index the index
     * @return expanded array
     */
    private int[] expandIfNeeded(int[] array, int index) {
```

```
        if (index >= array.length) {
            int[] temp = new int[array.length * 2];
            Arrays.fill(temp, -1);
            System.arraycopy(array, 0, temp, 0, array.length);
            array = temp;
        }
        return array;
    }

    /**
     * Expands the given array if the given index is out of bounds.
     *
     * @param array original array
     * @param index the index
     * @return expanded array
     */
    private char[] expandIfNeeded(char[] array, int index) {
        if (index >= array.length) {
            char[] temp = new char[array.length * 2];
            Arrays.fill(temp, (char) -1);
            System.arraycopy(array, 0, temp, 0, array.length);
            array = temp;
        }
        return array;
    }

    /**
     * Main function.
     * Master Control Component.
```

```
     *
     * @param args
     */
    public static void main(String[] args) {
        try {
            KWIC kwic = new KWIC();
            kwic.input(new FileReader(new File("input.txt")));
            kwic.circularShift();
            kwic.alphabetizer();
            kwic.output(new FileWriter(new File("output.txt")));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```