УДК 004.5

П. А. Курта
Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича

Взаимодействие пользователя с информационной системой. Часть 2. Алгоритмы обнаружения недостатков*

Статья является второй из цикла, посвященных вопросам взаимодействия пользователя с информационной системой. Решается задача обнаружения недостатков взаимодействий. Для этого вводится терминологическая база, определяющая следующие основные объекты процесса взаимодействия: элемент, форма, решение задачи, функция решения задачи, граф сценария, узел графа сценария, граф данных, узел графа данных, интерфейс, предметная область задачи; а также их взаимосвязь. С использованием базы выделяются отличительные признаки каждого из Топ-20 недостатков взаимодействия, определенных в первой части цикла. Как следствие, для каждого недостатка предлагается алгоритм его обнаружения, описанный на объектно-ориентированном псевдоязыке; дается описание принципа работы каждого из алгоритмов. Анализ алгоритмов позволил воссоздать общий программный каркас, на базе которого возможна модернизация текущих, а также создание новых способов поиска. Делается вывод касательно перехода от достаточно субъективного описания недостатков к значительно более строгому базису для их формализации. Формулируется направление дальнейшего исследования, которое будет освещено в заключительной статье цикла.

Информационная система, интерфейс, алгоритм обнаружения, терминологическая база, программный каркас

Одной из сторон существования человека в современном мире является его постоянное взаимодействие с окружающими информационными системами [1]–[4]. Успешность же существования зависит от эффективности такого взаимодействия. Факторами, негативно влияющими на эффективность, являются различные недостатки взаимодействий, Топ-20 которых был описан в 1-й части цикла авторских статей [5]. При этом, в сложных случаях и при наличии множества недостатков они могут оказывать взаимное влияние и друг на друга [6],]7]. Продолжая исследование, во 2-й части цикла будет предложена терминологическая база, определяющая взаимодействие пользователя и информационной системы, которая позволит в ее терминах определить отличительные признаки каждого из недостатков. Затем, решая задачу их обнаружения, на основании признаков и используя объектно-ориентированный псевдокод, будет описан обобщенный алгоритм поиска каждого недостатка. Систематизация объектов и используемых методов псевдокода позволит создать программный каркас, представляющий собой базовый набор программных классов [8] для создания полноценных алгоритмов определения характеристик недостатков и итоговой оценки эффективности взаимодействий.

Терминологическая база. Исходя из описания недостатков, сделанного в 1-й части цикла статей, можно выделить следующие элементы терминологической базы [9], [10], необходимой для формулирования признаков каждого из 20 недостатков взаимодействий:

1. *Элемент* – основной графический объект, через который происходит передача информации между пользователем и информационной системой; объект может быть различного типа (например, полем ввода/вывода, текстовой меткой и пр.), имеет расположение на Форме, характеристики (временные и когнитивные затраты, требования к работе с ним человека), может частично настраиваться пользователем, содержать собственное смысловое назначение, а также в ряде случаев ссылки на внешние ресурсы.

2. *Форма* – графический объект, содержащий Элементы и ассоциированный с Узлом графа сценария.

3. *Решение задачи* – объект, реализующий способ вычисления результата задачи, а также связанный с решением тезаурус [11], [12].

* Продолжение. Начало см. в [5].

4. *Функция Решения задачи* – главная функция объекта Решение задачи, вычисляющая результат, необходимый пользователю, по введенным им параметрам.

5. *Граф сценария* – ориентированный граф, в котором вершина соответствует некому состоянию работы пользователя при взаимодействии с информационной системой, а дуга – переходу из текущего состояния в следующее (аналогично графу потока управления, используемому при анализе кода программного обеспечения [13]); исходя из условий работы информационной системы по решению одной задачи граф имеет один начальный и один конечный узлы; также по графу возможно определение циклов и вычисление метрик сложности (аналогично графу потока данных [14]).

6. *Узел графа сценария* – объект-состояние работы пользователя, связанный с некоторой Формой; также узел может вызывать действия для Решения задачи.

7. *Граф данных* – ориентированный граф, где вершина соответствует некой переменной для вычисления результата Решения задачи, а дуга – зависимости между двумя переменными.

8. *Узел графа данных* – объект-переменная работы пользователя, связанный с некоторой внутренней переменной, Элементом, параметром или результатом Решения задачи.

9. *Интерфейс* – внешний объект взаимодействия, содержащий все Формы, Элементы, тезаурус и ряд вспомогательных объектов [15].

10. *Предметная область задачи* – множество предметов (статические объекты), которыми оперирует Решение задачи, а также их соотношения (динамические объекты).

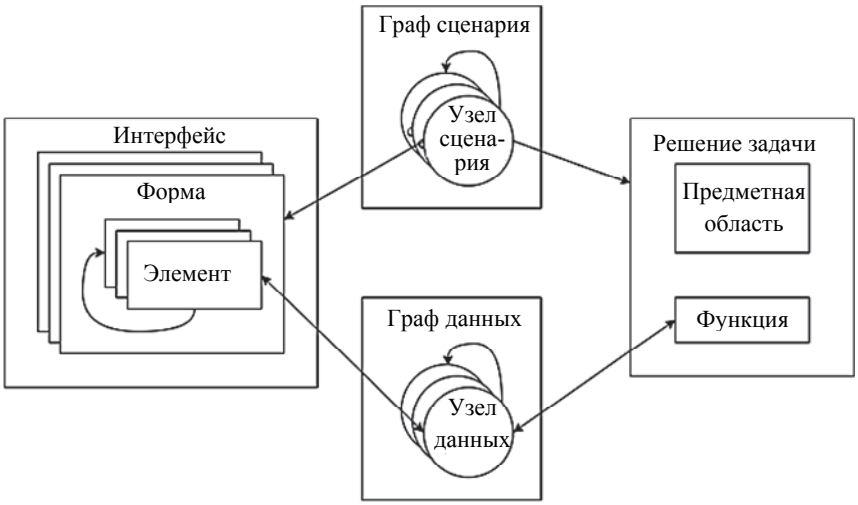
Взаимосвязь объектов приведенной терминологической базы показана на рисунке.

Алгоритмы обнаружения. Поскольку строгое определение и параметризация настолько неоднозначного понятия, как «недостаток», является отдельной сверхсложной задачей, то для его формализации применим следующий подход. Во-первых, для каждого недостатка будем указывать его отличительные признаки в терминах, совпадающих или являющихся частью приведенной терминологической базы. Во-вторых, будем приводить часть псевдокода (на условном объектно-ориентированном языке, близком к C#) [16], гипотетически предназначенного для обнаружения недостатков в конкретной реализации взаимодействия. При задании алгоритмов обнаружения будут использоваться программные классы (а также их экземпляры), методы и свойства, описание которых будет раскрыто далее – в программном каркасе. Псевдокод опишем в форме функций класса Interface вида *Has_Weakness_ No_i()*, возвращающих true при обнаружении недостатка с индексом *i*.

Недостаток 1. Недостаточность запрашиваемой информации. Отличительные признаки недостатка заключаются в следующем: «На Графе данных отсутствует путь между одним из параметров Функции Решения задачи и каким-либо Элементом ввода».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_1 () {
    foreach (var parameter in Deci-
sion.Parameters) {
        var f = false;
        foreach (var element in Inter-
face.Elements) {
            if (element.HasInput()) {
                if (Data-
Graph.HasPath(parameter, element)) {
```



```
                    f = true;
                    break;
                }
            }
        }
        if (!f)
            return true;
    }
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_1) вернет успех (true), если в графе данных (DataGraph) для каждого параметра (.Parameters) функции решения (Decision) не будет найдено логической связи (.HasPath) ни с одним элементом (.Elements) интерфейса (Interface), имеющим возможность ввода (.HasInput); в ином случае вернется неудача (false).

Недостаток 2. Избыточность запрашиваемой информации. Отличительные признаки недостатка заключаются в следующем: «На Графе данных отсутствует путь между Элементом ввода и каким-либо Параметром Функции Решения задачи».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_2 () {
    foreach (var element in Inter-
face.Elements) {
        if (!element.HasInput())
            continue;
        var f = false;
        foreach (var parameter in Deci-
sion.Parameters) {
            if (DataGraph.HasPath(element,
parameter)) {
              f = true;
                break;
            }
        }
        if (!f)
              return true;
    }
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_2) вернет успех (true), если в графе данных (DataGraph) для каждого из элементов (.Elements) интерфейса (Interface), имеющего возможность ввода (.HasInput), не будет найдено логической связи (.HasPath) ни с одним параметром (.Parameters) функции решения (Decision); в ином случае вернется неудача (false).

Недостаток 3. Тупиковый путь сценария. Отличительные признаки недостатка заключаются в следующем: «На Графе сценария присутствует не финальный Узел без выходных дуг».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_3 () {
    foreach (var node in ScenaryGraph.Nodes)
{
        if (node.IsFinal())
            continue;
        if (node.NumOfOutgoing() == 0)
            return true;
    }
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_3) вернет успех (true), если найдется узел из списка всех узлов (.Nodes) графа сценария (ScenaryGraph), не являющийся финальным (.IsFinal), число исходящих связей из которого (.NumOfOutgoing) равно 0; в ином случае вернется неудача (false).

Недостаток 4. Цикличность пути сценария. Отличительные признаки недостатка заключаются в следующем: «На Графе сценария доля циклов, состоящих более, чем из одного Узла, больше предельного значения».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_4 () {
    var loops = Sce-
naryGraph.FindLoops(MoreOneNode: true);
    var nodes = ScenaryGraph.Nodes;
    if (loops.Length / nodes.Length > Limits.
MaxSGRateOfLoopsWithN)
        return true;
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_4) вернет успех (true), если отношение числа циклов (.FindLoops) графа сценария (ScenaryGraph), имеющих больше одного узла (MoreOneNode: true), к числу всех узлов графа (.Nodes) больше соответствующей максимальной доли (.MaxSGRateOfLoopsWithN) из списка предустановленных пределов (Limits); в ином случае вернется неудача (false). Требование касательно наличия в цикле более чем одного узла связано с тем, что циклы из одного узла используются при обнаружении другого недостатка.

Недостаток 5. Сложность путей сценария. Отличительные признаки недостатка заключаются в следующем: «Компоненты метрики сложности Графа сценария больше предельных значений».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_5 () {
    var metric = ScenaryGraph.BuildMetric();
    foreach (var key in metric.Keys) {
        if (metric[key] > Lim-
its.MaxSGMetric[key])
            return true;
    }
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_5) вернет успех (true), если в метрике сложности (.BuildMetric), полученной из графа сценария (ScenaryGraph), одно из значений ее компонентов (.Keys) больше соответствующего максимального значения метрики (.MaxSGMetric) из списка предустановленных пределов (Limits); в ином случае вернется неудача (false).

Недостаток 6. Использование различных тезаурусов. Отличительные признаки недостатка заключаются в следующем: «Доля объектов Тезауруса Интерфейса, принадлежащих Тезаурусу Решения задачи, меньше предельного значения».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_6 () {
    var thsI = Interface.Thesaurus();
    var thsD = Decision.Thesaurus();
    if (Intersection(thsI, thsD).Length /
thsI.Length > Limits.MaxRateOfThesaurus)
        return true;
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_6) вернет успех (true), если отношение размера (.Length) множества, полученного пересечением (Intersection) тезауруса (.Thesaurus) интерфейса (Interface) и тезауруса решения задачи (Decision), к размеру тезауруса интерфейса больше соответствующей максимальной доли (.MaxRateOfThesaurus) из списка предустановленных пределов (Limits); в ином случае вернется неудача (false).

Недостаток 7. Непредсказуемость поведения. Отличительные признаки недостатка заключаются в следующем: «Вероятность появления определенного Элемента за текущим меньше предельного значения».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_7 () {
    var nodePairs = Sce-
naryGraph.BuildNodePairs();
    foreach (var element1 in node-
Pairs.Node1.ELements) {
        foreach (var element2 in node-
Pairs.Node2.ELements) {
            var prob = Lim-
its.ElementPairProbaility(element1, ele-
ment2);
            if (prob == null)
                continue;
            if (prob.Value < Lim-
its.MinElementPairProbailityValue)
                return true;
        }
    }
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_7) работает следующим образом. Во-первых, по графу сценария (ScenaryGraph) собираются все пары соседних узлов (.BuildNodePairs). Во-вторых, у каждого такого узла из пары (.Node1 и .Node2) получается список ассоциированных с ними элементов (.Elements). В-третьих, происходит полный перебор и выбор всех комбинаций элементов 1-го и 2-го узлов пары. В-четвертых, для каждой пары выбранных элементов, используя список предустановленных пределов (Limits), определяется статистическая (определенная заранее) вероятность встречи их подряд в типовых интерфейсах (.ElementPairProbaility). В-пятых, проверяется вероятность встречи каждого нового элемента за предыдущим при работе с текущим интерфейсом по сравнению с аналогичным, применяемым для решения подобного рода задач, что позволяет говорить о предсказуемости последовательности элементов для пользователя. Если такая запись с вероятностью отсутствует (prob = = null), то считается, что это нововведение в интерфейс, которое не может быть оценено на предмет предсказуемости; таким образом, данная пара элементов пропускается. Если же вероятность определена, то она проверяется на предмет недостижения допустимо минимальной (MinElementPairProbailityValue) из списка предустановленных пределов (Limits); в этом случае функция вернет успех (true). В ином случае – если вероятность встречи всех пар элементов оказалась выше минимальной – из функции вернется неудача (false).

Недостаток 8. Переполнение интерфейсных форм элементами. Отличительные признаки недостатка заключаются в следующем: «Количество и занимаемая площадь Элементов на одной из Форм больше предельного значения» [17].

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_8 () {
    foreach (var form in Interface.Forms) {
        if (form.Elements.Length > Lim-
its.MaxNumOfElementsAtForm)
            return true;
```

```
        foreach (var element1 in
form.Elements) {
            if (element1.ReversePos() > Lim-
its.MaxPosXOfElementAtForm)
                return true;
        }
    }
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_8) вернет успех (true) в двух случаях: 1) если для какой-либо из форм (.Forms) интерфейса (Interface) количество вложенных в нее Элементов (.Elements) больше максимального количества (.MaxNumOfElementsAtForm) из списка предустановленных пределов (Limits); 2) если правая нижняя позиция (.ReversePos) Элемента, расположенного внутри Формы, больше предельной позиции (.MaxPos OfElementAtForm) из списка предустановленных пределов. В ином случае из функции вернется неудача (false).

Недостаток 9. Использование времязатратных элементов. Отличительные признаки недостатка заключаются в следующем: «Временная затратность Элементов ввода и вывода данных, типовых для Решения задачи, больше предельного значения».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_9 () {
    foreach (var element in Inter-
face.Elements) {
        if (!element.HasInput() && !ele-
ment.HasOutput())
            continue;
        var time = ele-
ment.AverageOperationTime;
        if (time > Lim-
its.MaxDecisionOperationTime)
            return true;
    }
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_9) вернет успех (true), если для любого из элементов (.Elements) интерфейса (Interface), имеющего ввод (.HasInput) или вывод (.Has Output) данных, среднее время его работы (.AverageOperationTime) больше максимально допустимого времени работы элементов при решении задачи (.MaxDecisionOperationTime) из списка предустановленных пределов (Limits); в ином случае вернется неудача (false).

Недостаток 10. Высокая когнитивная нагрузка. Отличительные признаки недостатка заключаются в следующем: «Когнитивная нагрузка каждого Элемента, а также суммарная для Элементов Формы, больше предельной».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_10 () {
    foreach (var element in Inter-
face.Elements) {
        var cognitiveLoad = ele-
ment.CognitiveLoad;
        if (cognitiveLoad > Lim-
its.MaxElementCognitiveLoad)
            return true;
    }
    foreach (var form in Interface.Forms) {
        var sumCognitiveLoad = 0;
        foreach (var element1 in
form.Elements) {
            sumCognitiveLoad += ele-
ment1.CognitiveLoad;
        }
        if (sumCognitiveLoad > Lim-
its.MaxSumCognitiveLoad)
            return true;
    }
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_10) вернет успех (true) в двух случаях: 1) если для какого-либо из элементов (.Elements) интерфейса (Interface) его когнитивная нагрузка (.CognitiveLoad) больше максимального значения (.MaxCognitiveLoad) из списка предустановленных пределов (Limits); 2) если сумма когнитивных нагрузок элементов (.Elements) внутри каждой из форм (.Forms) больше максимального значения (.Max SumCognitiveLoad) из списка предустановленных пределов. В ином случае из функции вернется неудача (false).

Недостаток 11. Игнорирование ограничений организма человека. Отличительные признаки недостатка заключаются в следующем: «Характеристики Элементов, относящиеся к особенностям организма Пользователя, больше предельного значения».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_11 () {
    foreach (var element in Inter-
face.Elements) {
        var features = element.HumanFeatures;
        foreach (var key in features.Keys) {
```

```
            if (features[key] > Lim-
its.MaxHumanFeatures[key])
                return true;
        }
    }
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_11) вернет успех (true), если для какого-либо из элементов (.Elements) интерфейса (Interface), в связанных с ним человеческих особенностях (.HumanFeatures), одна из особенностей (.Keys) больше соответствующего максимального значения особенности (.MaxHumanFeatures) из списка предустановленных пределов (Limits); в ином случае вернется неудача (false).

Недостаток 12. Отсутствие ответной реакции на действия. Отличительные признаки недостатка заключаются в следующем: «На Графе сценария доля циклов составом из одного узла меньше предельного значения».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_12 () {
    var loops = Sce-
naryGraph.FindLoops(MoreOneNode: false);
    var nodes = ScenaryGraph.Nodes;
    if (loops.Length / nodes.Length < Lim-
its.MinSGRateOfLoopsWith1)
        return true;
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_12) вернет успех (true), если отношение числа циклов (.FindLoops) графа сценария (ScenaryGraph), имеющих один узел (MoreOneNode: false), к числу всех узлов графа (.Nodes) меньше соответствующей минимальной доли (.MinSGRateOfLoopsWith1) из списка предустановленных пределов (Limits); в ином случае вернется неудача (false). Требование касательно наличия в цикле одного узла связано с тем, что циклы больше, чем из одного узла, используются при обнаружении другого недостатка.

Недостаток 13. Слабая адаптируемость под пользователя. Отличительные признаки недостатка заключаются в следующем: «Среднее количество Настроек Элементов, доступных пользователю, меньше предельного значения».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_13 () {
    var num = 0;
    foreach (var element in Inter-
face.Elements) {
        var options = element.Options;
        num += options.Length;
    }
    if (num / Interface.Elements.Length <
Limits.MinAverageNumOfOptions)
        return true;
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_13) вернет успех (true), если отношение суммарного количества настроек (.Options) каждого из элементов (.Elements) интерфейса (Interface) к количеству всех элементов (.Length) меньше минимального числа (.MinAverage NumOfOptions) из списка предустановленных пределов (Limits); в ином случае вернется неудача (false).

Недостаток 14. Неоднозначность трактовки элементов. Отличительные признаки недостатка заключаются в следующем: «Вероятность появления Элементов, имеющих определенные Действия, меньше предельного значения».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_14 () {
    foreach (var element in Inter-
face.Elements) {
        foreach (var action in ele-
ment.Actions) {
            var prob = Lim-
its.ElementToActionProbaility(element, ac-
tion);
            if (prob == null)
                continue;
            if (prob.Value < Lim-
its.MinElementToActionProbailityValue)
                return true;
        }
    }
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_14) работает следующим образом. Во-первых, для каждого из элементов (.Elements) интерфейса (Interface) получается список ассоциированных с ним действий (.Actions). Во-вторых, для каждой пары выбранного элемента и одного из действий, используя список предустановленных пределов (Limits), определяется статистическая (определенная заранее) вероятность встречи их подряд в типовых интерфейсах (.ElementToActionProbability). В-третьих, проверяется вероятность встречи элемента в текущем интерфейсе, который может приводить к определенному действию, в анало-

гичном, что позволяет говорить о предсказуемости поведения элемента для пользователя. Если такая запись с вероятностью отсутствует (prob = = null), то считается, что это нововведение в интерфейс, которое не может быть оценено на предмет предсказуемости; таким образом, данная пара пропускается. Если же вероятность определена, то она проверяется на предмет недостижения допустимо минимальной (MinElementToActionProbailityValue) из списка предустановленных пределов; в этом случае функция вернет успех (true). В ином случае – если вероятность встречи пар элементов и действий оказалась выше или равной допустимой – из функции вернется неудача (false).

Недостаток 15. Отсутствие ориентированности элементов на предметную область. Отличительные признаки недостатка заключаются в следующем: «Доля смысловых назначений всех Элементов, принадлежащих множеству статических объектов Предметной области Решения задачи, меньше предельного значения».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_15 () {
    HashSet set;
    foreach (var element in Inter-
face.Elements) {
       var proposal = element.Proposal;
       set.Add(proposal)
    }
    if (Intersection(set, Deci-
sion.SubjectArea.StaticObjectsSet).Length /
set.Length < Lim-
its.MinSubjectAreaStaticProposal)
        return true;
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_15) вернет успех (true), если отношение размера (.Length) множества, полученного пересечением (Intersection) между смысловыми назначениями (.Proposal) всех элементов (.Elements) интерфейса (Interface) и статическими объектами (.StaticObject) предметной области (.SubjectArea) решения задачи (Decision), к размеру множества смысловых значений всех элементов меньше соответствующей минимальной доли (.MinAverageNumOfOptions) из списка предустановленных пределов (Limits); в ином случае вернется неудача (false).

Недостаток 16. Отсутствие ориентированности сценария на предметную область. Отличительные признаки недостатка заключаются в следующем: «Доля смысловых назначений всех Действий Элементов и Действий Узлов Графа сценария, принадлежащих множеству динамических объектов Предметной области Решения задачи, меньше предельного значения».

Деление объектов Предметной области на статические и динамические связано с тем, что первые определяют набор понятий области, а вторые – их отношения (т. е. смысловую взаимосвязь).

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_16 () {
    HashSet set;
    foreach (var element in Inter-
face.Elements) {
        var proposal = ele-
ment.Actions.Proposals;
        set.Add(proposal)
    }
    foreach (var actions in Sce-
naryGraph.Actions) {
        var proposal = actions.Proposals;
        set.Add(proposal)
    }
    if (Intersection(set, Deci-
sion.SubjectArea.DynamicObjectsSet).Length /
set.Length < Lim-
its.MinSubjectAreaDynamicProposal)
        return true;
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_16) вернет успех (true), если отношение размера (.Length) множества, полученного пересечением (Intersection) между смысловыми назначениями (.Proposals) действий (.Actions) всех элементов (.Elements) интерфейса (Interface), а также действий (.Actions) всех узлов графа сценария (ScenaryGraph), и динамическими объектами (.StaticObject) предметной области (.Subject Area) решения задачи (Decision), к размеру множества смысловых значений всех действий элементов меньше соответствующей минимальной доли (.MinSubjectAreaDynamicProposal) из списка предустановленных пределов (Limits); в ином случае вернется неудача (false).

Недостаток 17. Нехватка помощи пользователю. Отличительные признаки недостатка заключаются в следующем: «Доля Элементов ввода и вывода данных, перед которыми расположена текстовая метка с помощью, меньше предельного значения».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_17 () {
    var num = 0;
    foreach (var element in Inter-
face.Elements) {
```

```
        if (!element.IsLabel())
            continue;
        var next = element.Next();
        if (!next.HasInput() &&
!next.HasOutput())
            continue;
        ++num;
    }
    if (num / Interface.Elements.Length <
        Limits.MinIOElementsWithPrevLabel)
        return true;
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_17) вернет успех (true), если отношение количества элементов (.Elements) интерфейса (Interface), имеющих ввод (.HasInput) или вывод (.HasOutput) данных, которые следуют (.Next) за метками (.IsLabal), к числу всех элементов (.Length) меньше минимального числа (.MinIO ElementsWithPrevLabel) из списка предустановленных пределов (Limits); в ином случае вернется неудача (false).

Недостаток 18. Возможность скрытых информационных воздействий через элементы. Отличительные признаки недостатка заключаются в следующем: «Параметры Элементов содержат ссылки на внешние ресурсы» [18], [19].

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_18 () {
    foreach (var element in Inter-
face.Elements) {
        var properties = element.Properties;
        foreach (var key in properties.Keys)
{
            if (HasExternal-
Link(properties[key]))
                return true;
        }
    }
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_18) вернет успех (true), если для какого-либо из элементов (.Elements) интерфейса (Interface), в связанных с ним параметрах (.Properties), один из них (.Keys) будет содержать ссылку на внешний ресурс (HasExternalLink); в ином случае вернется неудача (false).

Недостаток 19. Возможность скрытых информационных воздействий через сценарий. Отличительные признаки недостатка заключаются в следующем: «На Графе данных доля Элементов без связи с Параметрами и Результатом функции Решения задачи больше критической» [20].

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_19 () {
    var num = 0;
    foreach (var element in Inter-
face.Elements) {
        var f = false;
        foreach (var parameter in Deci-
sion.Parameters) {
            if (DataGraph.HasPath(element,
parameter)) {
                f = true;
                break;
            }
        }
        if (DataGraph.HasPath(element, Deci-
sion.Result))
            f = true;
        if (f)
            ++num;
    }
    if (num / Interface.Elements.Length >
Limits.MaxNumOfElementsWithoutLinkToFunction)
        return true;
    return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_19) работает следующим образом. Вначале вычисляется количество элементов (Elements) интерфейса (Interface), которые по Графу данных (DataGraph) логически связаны (HasPath) или с одним из параметров (.Parameters) функции решения (Decision), или с ее результатом (.Result). Затем, если доля найденных таким образом элементов в списке всех элементов больше соответствующей максимальной доли (.MaxNumOfElementsWithoutLinkToFunction) из списка предустановленных пределов (Limits), то функция вернет успех (true); в ином случае – неудача (false).

Недостаток 20. Дублирование вывода результатов. Отличительные признаки недостатка заключаются в следующем: «На Графе данных путь из Результата функции Решения задачи доходит до двух Элементов с выводом данных».

Обнаружить недостаток можно с помощью следующего псевдокода:

```
bool Interface.Has_Weakness_No_20 () {
    var num = 0;
    foreach (var element in Inter-
face.Elements) {
        if (!element.HasOutput())
            continue;
        if (DataGraph.HasPath(element, Deci-
sion.Result))
            ++num;
    }
    if (num > 1)
        return true;
```

```
    else
        return false;
}
```

Функция псевдокода (Interface.Has_Weakness_No_20) вернет успех (true), если в графе данных (DataGraph) найдется больше одного пути (.HasPath) между элементами (.Elements) интерфейса (Interface), имеющего возможность вывода (.HasOutput), и результатом (.Result) функции решения (Decision); в ином случае вернется неудача (false).

Программный каркас. Исходя из предложенного объектно-ориентированного псевдокода (который представляет собой описание алгоритмов обнаружения недостатков во взаимодействии) составим программный каркас – по сути, обобщенную модель программы, на базе которого возможна реализация этих алгоритмов – частную модель программы (в том смысле, что она строится и уточняется по обобщенной, аналогично описанным для сферы информационной безопасности [21]–[23]). Это позволит как расширять существующие, так и создавать новые алгоритмы, делая минимальные изменения в каркасе. Последний можно представить в табличном виде, где основные строки – программные классы, а их подстроки – методы и свойства класса (таблица).

Класс	**Метод или свойство**	**Описание**
Interface		Класс, ассоциируемый с Интерфейсом
	Elements	Список Элементов
	Forms	Список Форм
	Thesaurus()	Сборка тезауруса
Decision		Класс, ассоциируемый с Решением задачи
	Parameters	Параметры функции
	Result	Результат функции
	SubjectArea	Предметная область (класс SubjectArea)
	Thesaurus()	Сборка тезауруса (которую возможно осуществить в автоматическом режиме [24])
SubjectArea		Класс, ассоциируемый с Предметной областью
	StaticObjectsSet	Список предметов, которыми оперирует Решение задачи
	DynamicObjectsSet	Список соотношений предметов, которыми оперирует Решение задачи
ScenaryGraph		Класс, ассоциируемый с Графом сценария
	Nodes	Список узлов
	Actions	Список действий узлов
	FindLoops()	Поиск циклов графа (реализация которого имеет достаточное количество решений [25], [26])
	BuildMetric()	Построение метрики сложности графа (имеющей множество вариаций [27]–[29])
	BuildNodePairs()	Построение списка пар связанных узлов
DataGraph		Класс, ассоциируемый с Графом данных
	HasPath()	Проверка по графу связи между узлами
Element		Класс, ассоциируемый с графическим объектом Интерфейса – Элементом
	HasInput()	Проверка наличия у Элемента ввода данных
	HasOutput()	Проверка наличия у Элемента вывода данных
	IsLabel()	Проверка на соответствие Элемента текстовой метке [30]
	ReversePos()	Получение обратных координат Элемента на Форме (т. е. противоположных от начального угла его области)
	Next()	Получение следующего Элемента на Форме
	AverageOperation Time	Среднее время работы с Элементом
	CognitiveLoad	Когнитивная нагрузка при работе с Элементом [31]
	HumanFeatures	Особенности человека, необходимые для работы с Элементом
	Options	Настройки, доступные пользователю
	Actions	Действия, ассоциированные с Элементом
	Proposal	Смысловое назначение Элемента
	Properties	Параметры Элемента (включая ссылку на внешний ресурс)
Form		Класс, ассоциируемый с графическим объектом Интерфейса – Формой
	Elements	Список Элементов, находящихся на Форме
Limits		Класс, содержащий критерии, необходимые для выявления недостатков взаимодействий
	...	Предельные константы и распределения

В данной части цикла статей введен терминологический базис, который позволил в его объектах частично формализовать признаки недостатков взаимодействий пользователя с информационной системой, выделенных в 1-й части цикла. Такая форма записи признаков позволила предложить алгоритмы обнаружения каждого из недостатков, используя единый программный каркас из набора классов, их методов и свойств. Таким образом, удалось перейти от достаточно субъективного описания недостатков к значительно более строгому базису для их формализации и разработки алгоритмов поиска. Последний позволит синтезировать подход к вычислению различных характеристик недостатков (помимо простого установления факта их наличия), что будет востребовано при создании методики оценки компонентов эффективности такого взаимодействия (например, аналогично [32]–[34]). Автор планирует сделать это в 3-й заключительной части цикла статей.

СПИСОК ЛИТЕРАТУРЫ

1. Израилов К. Е., Покусов В. В. Актуальные вопросы взаимодействия элементов комплексных систем защиты информации // Актуальные проблемы инфотелекоммуникаций в науке и образовании (АПИНО 2017): сб. науч. статей VI Междунар. науч.-техн. и науч.-метод. конф., СПб., 2017. С. 255–260.

2. Digital cloud environment: present challenges and future forecast / S. Mescheryakov, D. Shchemelinin, K. Izrailov, V. Pokussov // Future Internet. 2020. Т. 12, № 5. P. 82.

3. Покусов В. В. Особенности взаимодействия служб обеспечения функционирования информационной системы // Информатизация и связь. 2018. № 5. С. 51–56.

4. Research of the mobile cdma network for the operation of an intelligent information system of earth-moving and construction machines / T. Golubeva, S. Konshin, E. Zaytcev, V. Pokussov, B. Tshukin // Lecture Notes in Computer Science. 2017. Т. 10486 LNCS. P. 193–205.

5. Курта П. В. Взаимодействие пользователя с информационной системой. Ч. 1. Схема взаимодействия и классификация недостатков // Изв. СПбГЭТУ «ЛЭТИ». 2020. № 8–9. С. 35–45.

6. Буйневич М. В., Израилов К. Е. Антропоморфический подход к описанию взаимодействия уязвимостей в программном коде. Ч. 1. Типы взаимодействий // Защита информации. Инсайд. 2019. № 5 (89). С. 78–85.

7. Буйневич М. В., Израилов К. Е. Антропоморфический подход к описанию взаимодействия уязвимостей в программном коде. Ч. 2. Метрика уязвимостей // Защита информации. Инсайд. 2019. № 6 (90). С. 61–65.

8. Повар А. В., Прохоров П. В. Построение программных каркасов для специального программного обеспечения // Сб. докл. Междунар. науч.-техн. конф. РАДИОТЕХНИКА, ЭЛЕКТРОНИКА И СВЯЗЬ («РЭИС-2011») / Омский науч.-исслед. ин-т приборостроения. Омск, 2011. С. 262–265.

9. Моисеенко Г. Е. Слова, слова, слова (о терминологических базах) // Мосты. Журн. переводчиков. 2012. № 1 (33). С. 26–30.

10. Перкова О. В. Использование терминологической базы данных при обучении языку для специальных целей // Новые технологии в образовательном пространстве родного и иностранного языка. 2013. № 1. С. 77–82.

11. Маркелов К. С., Нейман А. Б. Тезаурус предметных областей // Альманах современной науки и образования. 2012. № 4. С. 154–155.

12. Земляная Т. Б., Павлычева О. Н. Терминологические словари и тезаурусы // Журн. науч.-пед. информации. 2010. № 4. С. 1–29.

13. Клименко В. Ю., Амосов В. В. Анализ циклов в графах потока управления // Сб. материалов конф. «Современные технологии в теории и практике программирования» СПб.: Изд-во С.-Петерб. политехн. ун-та Петра Великого, 2019. С. 111–113.

14. Кузнецов С. В., Мазин А. В. Обфускация графа потока данных // Вопросы радиоэлектроники. 2010. Т. 3, № 4. С. 63–67.

15. Грибова В. В., Тарасов А. В. Модель онтологии предметной области «графический пользовательский интерфейс» // Информатика и системы управления. 2005. № 1 (9). С. 80–90.

16. Журов Д. П., Пятых С. О., Сосинская С. С. Использование внешнего DSL Antlr для разработки программного обеспечения построения блок–схем по псевдокоду // Austrian J. of Technical and Natural Sciences. 2015. № 5–6. P. 49–50.

17. Длясин Д. Г. Проблемы пространственно–плоскостного взаимодействия изображения и шрифта в графическом дизайне // Изв. Самарского науч. центра Российской акад. наук. 2009. Т. 11, № 4–4. С. 1064–1067.

18. Гращенко Л. А. Скрытые средства воздействия на массовое сознание в текстах СМИ // Вестн. Российского нового ун-та. Сер.: Человек в современном мире. 2019. № 3. С. 86–91.

19. Сердюкова Е. Ф. Приемы скрытого управляющего воздействия в информационных материалах экстремистского толка, распространяемых в сети Интернет // Вестн. Костромского гос. ун-та. Сер.: Педагогика. Психология. Социокинетика. 2018. Т. 24, № 2. С. 65–67.

20. Ковалева И. Ю. Предотвращение угрозы информационной безопасности населения в области рекламы // Вестн. УрФО. Безопасность в информационной сфере. 2013. № 4 (10). С. 12–16.

21. Архитектурные уязвимости моделей телекоммуникационных сетей / М. В. Буйневич, О. В. Щербаков, А. Г. Владыко, К. Е. Израилов // Науч.–аналитический журн. Вестн. С.-Петерб. ун-та Гос. противопожарной службы МЧС России. 2015. № 4. С. 86–93.

22. Буйневич М. В., Щербаков О. В., Израилов К. Е. Структурная модель машинного кода, специализированная для поиска уязвимостей в программном обеспечении автоматизированных систем управления // Проблемы управления рисками в техносфере. 2014. № 3 (31). С. 68–74.

23. Буйневич М. В., Израилов К. Е., Щербаков О. В. Модель машинного кода, специализированная для поиска уязвимостей // Вестн. Воронежского ин-та ГПС МЧС России. 2014. № 2 (11). С. 46–51.

24. Кузнецов Л. А., Капнин А. В. Технология автоматического формирования тезауруса русского языка // Информационные системы и технологии. 2012. № 4 (72). С. 14–19.

25. Клименко В. Ю., Сараджишвили С. Э. Оптимизация поиска циклов в графах потока управления // Велес. 2019. № 10–1 (76). С. 63–66.

26. Пьянков О. В. Разработка и исследование алгоритмов параллельного поиска циклов в графе // Вестн. Воронежского ин-та МВД России. 2015. № 1. С. 54–61.

27. Усовершенствование метрик оценки сложности программ, основывающихся на анализе графа потока управления / А. М. Бабичев, В. Е. Подольский, А. Н. Грибков, С. С. Толстых, С. Г. Толстых // Вестн. Тамбовского гос. техн. ун-та. 2016. Т. 22, № 2. С. 208–216.

28. Воробьев А. В. Метрики как средство оценивания качества программного обеспечения // Авиакосмическое приборостроение. 2007. № 1. С. 19–31.

29. Царевский А. В. Априорная оценка алгоритмов обработки информации по топологическим метрикам сложности // Автоматизация процессов управления. 2012. № 1. С. 95–101.

30. Семенов А. А., Афанасьев Г. И. Технология создания графического интерфейса с помощью QML // Теория и практика современной науки. 2017. № 4 (22). С. 1052–1059.

31. Ведерников А. В., Бакаев М. А. Влияние опыта взаимодействия с интерфейсом на когнитивную нагрузку пользователя // Сб. науч. тр. конф. «Наука. Технологии. Инновации» (Новосибирск). 2016. С. 10–11.

32. Покусов В. В. Оценка эффективности системы обеспечения ИБ. Ч. 1. Показатели и модели представления // Защита информации. Инсайд. 2019. № 2 (86). С. 54–60.

33. Покусов В. В. Оценка эффективности системы обеспечения ИБ. Ч. 2. Методика и результаты // Защита информации. Инсайд. 2019. № 3 (87). С. 64–72.

34. Израилов К. Е. Методика оценки эффективности средств алгоритмизации, используемых для поиска уязвимостей // Информатизация и связь. 2014. № 3. С. 39–42.

P. A. Kurta

The Bonch-Bruevich Saint-Petersburg State University of Telecommunications

INTERACTION OF THE USER WITH THE INFORMATION SYSTEM. PART 2. ALGORITHMS FOR DETECTING DISADVANTAGES

The article is the second in a series devoted to the issues of user interaction with the information system. The problem of detecting interaction disadvantages is being solved. For this, a terminological base is introduced that defines the following main objects of the interaction process: element, form, problem solution, problem solving function, script graph, script graph node, data graph, data graph node, interface, problem domain; as well as their relationship. Using the database, the distinguishing features of each of the Top 20 interaction disadvantages identified in the first part of the cycle are highlighted. As a consequence, for each disadvantages , an algorithm for its detection is proposed, described in an object-oriented pseudo-language; the description of the principle of operation of each of the algorithms is given. The analysis of the algorithms made it possible to recreate a general software framework, on the basis of which it is possible to modernize the current ones, as well as create new search methods. A conclusion is drawn regarding the transition from a rather subjective description of the disadvantages to a much more strict basis for their formalization. The direction of further research is set, which will be covered in the next article of the cycle.

Information system, interface, detection algorithm, terminology base, software framework