#### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО

#### ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

**«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

#### ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КУРСОВОЙ ПРОЕКТ

МАКЕТНАЯ ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ДИАЛОГОВО-ТЕХНОЛОГИЧЕСКОГО КОМПЛЕКА ГЕНЕРАЦИИ СЛОЖНЫХ ПРОГРАММНЫХ СИСТЕМ

ПРИМЕР РЕАЛИЗАЦИИ МАКЕТА НА ОСНОВЕ АВТОМАТА МИЛИ

Выполнила: студентка 2 курса

гр.09-411 Рябова Д.А.

Проверил: старший преподаватель кафедры технологий программирования

Георгиев В.О.

Казань

2015

Содержание

[Введение 3](#_Toc438865673)

[Раздел I. Формальное представление модели диалогового интерфейса, построенного на основе автомата Мили 5](#_Toc438865674)

[Раздел II. Постановка технического задания для программной реализации модели 10](#_Toc438865675)

[Стандарты ГОСТ/ISO 14](#_Toc438865676)

[Раздел III. Представление результатов работы, полученных в ходе разработке программного обеспечения 15](#_Toc438865677)

[Раздел IV. Литература и источники 24](#_Toc438865678)

Введение

*Постановка, проблематика и задач курсовой работы*

На сегодняшний день существует множество видов моделей для конструирования программных систем. Каждая из них используется в определённой предметной области. Одной из таких моделей является модель, основанная на принципах автомата Мили. Основной для построения макета диалогового технологического комплекса генерации сложных программных систем в данной курсовой работе используется именно эта модель.

Автоматная модель является одной из самых доступных для понимания моделей, используемых для реализации программных систем. Благодаря её простоте и вариативности можно сконструировать не только отдельную систему, но и целый комплекс, который может служить базой для построения других сложных систем.

Цель курсовой работы:

1. Приобретение навыков в формировании технического задания по требованиям заказчика;
2. Самостоятельная разработка макета диалогового технологического комплекса генерации сложных программных систем в соответствии с требованиями технического задания;
3. Построение генератора сценариев для диалоговых систем;
4. Приобретение навыков оформления программной документации для дальнейшей эксплуатации проекта.

*Основные понятия и определения*

**Формальная грамматика** или просто **грамматика** в теории формальных языков — способ описания формального языка, то есть выделения некоторого подмножества из множества всех слов некоторого конечного алфавита. Различают порождающие и распознающие (или аналитические) грамматики — первые задают правила, с помощью которых можно построить любое слово языка, а вторые позволяют по данному слову определить, входит ли оно в язык или нет.

**Конечный автомат** — [абстрактный автомат](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82), число возможных внутренних состояний которого [конечно](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE).

**Автомат Мили** — конечный автомат, выходная последовательность которого зависит от состояния автомата и входных сигналов. Это означает, что в графе состояний каждому ребру соответствует некоторое значение (выходной символ). В вершины графа автомата Мили записываются выходящие сигналы, а дугам графа приписывают условие перехода из одного состояния в другое, а также входящие сигналы.

**Формальный язык** в математической логике и информатике — множество конечных слов (строк, цепочек) над конечным алфавитом. Понятие языка чаще всего используется в теории автоматов, теории вычислимости и теории алгоритмов. Научная теория, которая имеет дело с этим объектом, называется теорией формальных языков.

**Теория автоматов** — раздел дискретной математики, изучающий абстрактные автоматы — вычислительные машины, представленные в виде математических моделей — и задачи, которые они могут решать.

**Вложенный автомат стека** - конечный автомат, который может использовать стек, содержащий данные, которые могут быть дополнительными стеками.

**Граф зависимостей** — ориентированный граф, отображающий соотношение элементов некоторой совокупности в соответствии с выбранным транзитивным отношением над ней.

**Нагруженный граф** − ориентированный граф D = (V, X), на множестве дуг которого определена некоторая функция, которую называют весовой функцией.

Раздел I. Формальное представление модели диалогового интерфейса, построенного на основе автомата Мили

*Конкретизация теории автоматов*

**Абстрактный автомат** (в [теории алгоритмов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2)) — [математическая](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) [абстракция](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%86%D0%B8%D1%8F), [модель](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) [дискретного устройства](https://ru.wikipedia.org/w/index.php?title=%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE&action=edit&redlink=1), имеющего один вход, один выход и в каждый момент времени находящегося в одном [состоянии](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5) из множества возможных. На вход этому устройству поступают [символы](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB) одного [алфавита](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29), на выходе оно выдаёт символы (в общем случае) другого алфавита.

Формально абстрактный автомат определяется как пятерка:

\boldsymbol{A = (S, X , Y, \delta , \lambda)}

Где S — конечное множество [состояний](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5) автомата, X, Y — конечные входной и выходной алфавиты соответственно, из которых формируются строки, считываемые и выдаваемые автоматом, \delta : S \times X \rightarrow S— функция переходов, \lambda : S \times X \rightarrow Y— функция выходов.

Абстрактный автомат с выделенным начальным [состоянием](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5) называется инициальным автоматом. Таким образом, абстрактный автомат определяет семейство инициальных автоматов:

\boldsymbol{(s_i, A), s_i \in S}

Если функции переходов и выходов однозначно определены для каждой пары \boldsymbol{(s, x) \in S \times X}, то автомат называют детерминированным. В противном случае автомат называют недетерминированным или частично определенным.

Если функция переходов и/или функция выходов являются случайными, то автомат называют [вероятностным](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%BD%D1%8B%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82).

Ограничение числа параметров абстрактного автомата определило такое понятие как [конечный автомат](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D1%8B%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82).

Функционирование автомата состоит в порождении двух последовательностей: последовательности очередных состояний автомата \boldsymbol{s_1[1]s_2[2]s_3[3]...}и последовательности выходных символов \boldsymbol{y_1[1]y_2[2]y_3[3]...}, которые для последовательности символов \boldsymbol{x_1[1]x_2[2]x_3[3]...} разворачиваются в моменты дискретного времени t = 1, 2, 3, … Моменты дискретного времени получили название тактов.

Функционирование автомата в дискретные моменты времени t может быть описано системой рекуррентных соотношений: \boldsymbol{s(t+1)=\delta(s(t),x(t));}

\boldsymbol{y(t)=\lambda(s(t),x(t)).}

Для уточнения свойств абстрактных автоматов введена [классификация](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_%D0%B0%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D1%85_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%BE%D0%B2).

Абстрактные автоматы образуют фундаментальный класс дискретных моделей как самостоятельная модель, и как основная компонента [машин Тьюринга](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3%D0%B0), [автоматов с магазинной памятью](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82_%D1%81_%D0%BC%D0%B0%D0%B3%D0%B0%D0%B7%D0%B8%D0%BD%D0%BD%D0%BE%D0%B9_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C%D1%8E), [конечных автоматов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D1%8B%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82) и других преобразователей информации.

[Модель](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) абстрактного автомата широко используется, как базовая, для построения дискретных моделей автоматов, распознающих, порождающих и преобразующих последовательности [символов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB).

**Конечный автомат** — [абстрактный автомат](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82), число возможных внутренних состояний которого [конечно](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE).

Существуют различные способы задания алгоритма функционирования конечного автомата. Например, конечный автомат может быть задан в виде упорядоченной пятерки элементов некоторых [множеств](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE_%28%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29):

**~M = (V, Q, q_0, F, \delta) ,**

где

* V — *входной алфавит* (конечное множество *входных символов*), из которого формируются *входные слова*, воспринимаемые конечным автоматом;
* Q — *множество внутренних состояний*;
* q_0 — *начальное состояние* (q_0 \in Q);
* F — множество *заключительных, или конечных состояний* (F \subset Q);
* \delta — *функция переходов*, определенная как отображение \delta : Q \times ( V \cup \{ \lambda \} ) \rightarrow Q, такое, что \delta(q,a) = \{ r : q \rightarrow_a r \}, то есть значение функции переходов на упорядоченной паре (состояние, входной символ или пустая цепочка) есть множество всех состояний, в которые из данного состояния возможен переход по данному входному символу или пустой цепочке (λ).

Принято полагать, что конечный автомат начинает работу в состоянии *q0*, последовательно считывая по одному символу входного слова (цепочки входных символов). Считанный символ переводит автомат в новое состояние в соответствии с функцией переходов.

Читая входную цепочку символов *x* и делая переходы из состояния в состояние, автомат после прочтения последнего символа входного слова окажется в некотором состоянии *q'*.

Если это состояние является заключительным, то говорят, что автомат допустил слово *x*. Конечные автоматы широко используются на практике, например, в [синтаксических](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%82%D0%BE%D1%80) и [лексических анализаторах](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%82%D0%BE%D1%80), [тестировании программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) [на основе моделей](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BD%D0%B0_%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%B5_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8).

**Другие способы описания:**

* [**Диаграмма состояний**](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D0%9C%D1%83%D1%80%D0%B0) **(**или иногда **граф переходов**) — графическое представление множества состояний и функции переходов. Представляет собой размеченный ориентированный [граф](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_%28%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29), вершины которого — состояния КА, дуги — переходы из одного состояния в другое, а *метки дуг* — символы, по которым осуществляется переход из одного состояния в другое. Если переход из состояния *q1* в *q2* может быть осуществлен по одному из *нескольких* символов, то все они должны быть надписаны над дугой диаграммы.
* **Таблица переходов** — табличное представление функции *δ*. Обычно в такой таблице каждой строке соответствует одно состояние, а столбцу — один допустимый входной символ. В ячейке на пересечении строки и столбца записывается состояние, в которое должен перейти автомат, если в данном состоянии он считал данный входной символ

Конечные автоматы подразделяются на детерминированные и недетерминированные.

Детерминированным конечным автоматом (ДКА) называется такой автомат, в котором нет дуг с меткой *ε* (предложение, не содержащее ни одного символа), и из любого состояния по любому символу возможен переход в точности в одно состояние.

Недетерминированный конечный автомат (НКА) является обобщением детерминированного. Если рассмотреть случай, когда автомат задан следующим образом: ~M = (V, Q, S, F, \delta), где S — множество *начальных состояний* автомата, такое, что S \subseteq V, то появляется третий признак недетерминированности — наличие нескольких начальных (стартовых) состояний у автомата ~M.

*Теорема о детерминизации* утверждает, что для любого конечного автомата может быть построен эквивалентный ему детерминированный конечный автомат (два конечных автомата называют эквивалентными, если их языки совпадают). Однако поскольку количество состояний в эквивалентном ДКА в худшем случае растёт [экспоненциально](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D1%81%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D1%80%D0%BE%D1%81%D1%82) с ростом количества состояний исходного НКА, на практике подобная детерминизация не всегда возможна. Кроме того, [конечные автоматы с выходом](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D1%8B%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82_%D1%81_%D0%B2%D1%8B%D1%85%D0%BE%D0%B4%D0%BE%D0%BC) в общем случае не поддаются детерминизации.

В силу последних двух замечаний, несмотря на бо́льшую сложность недетерминированных конечных автоматов, для задач, связанных с обработкой текста, преимущественно применяются именно НКА.

**Автомат Мили** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Mealy machine*) — [конечный автомат](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D1%8B%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82), выходная последовательность которого (в отличие от [автомата Мура](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82_%D0%9C%D1%83%D1%80%D0%B0)) зависит от состояния автомата и входных сигналов. Это означает, что в графе состояний каждому ребру соответствует некоторое значение (выходной символ). В вершины графа автомата Мили записываются выходящие сигналы, а дугам графа приписывают условие перехода из одного состояния в другое, а также входящие сигналы.

Автомат Мили — совокупность \boldsymbol{A = (S, X, Y,\delta,\lambda,S_0)}, где

* S— [конечное непустое множество](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE) состояний автомата;
* X— конечное непустое множество входных символов;
* Y— конечное непустое множество выходных символов;
* \delta : S \times X \rightarrow S— функция переходов, отображающая пары состояние/входной символ на соответствующее следующее состояние;
* \lambda : S \times X \rightarrow Y— функция выходов, отображающая пары состояние/входной символ на соответствующий выходной символ;
* S_0\in S— начальное состояние.

**Кодировка автомата Мили:**

Вершина (операторная или логическая), стоящая после вершины "Начало", а также вход вершины "Конец" помечается символом S1, вершины, стоящие после операторных помечаются символом Sn (n=2,3..).

**Матрица функций переходов:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S/ X** | **C_1** | **C_2** | **C_3** |
| q1 | q1 / S | q2 / U1 | q3 / U2 |
| q2 | q1 / D1 | q2 / S | q3 / U1 |
| q3 | q1 / D2 | q2 / D1 | q3 / S |

#### Легенда:

* C_i— Входные символы;
* q_i— Внутренние состояния
* U_i, D_i, S— Выходные символы.
* q_i/ S— функция перехода

*Реализация модели для интерактивного взаимодействия в представленном диалоговом технологическом комплексе*

Структуру комплекса составляют две части: реализация модели с заданными свойствами с помощью, так называемого, генератора сценария и применение построенного сценария в диалоговой системе.

В качестве прототипа практической реализации модели для обеспечения интерактивного взаимодействия в макетном варианте выше представленная модель будет выглядеть следующим образом:

* Граф задается как пара {Nodes, Arrows}, что является стандартным представлением вершин графа и возможных переходов в следующие состояния;
* Каждой вершине соответствует вопрос диалога;
* Каждому переходу соответствует один из возможных вариантов ответов на вопрос, что по сути предоставляет нам возможность перемещаться в любые доступные состояния;
* Каждому вопросу, аналогично вершине графа, присваивается некоторое состояние.

Таким образом была реализована модель на основе принципов автомата Мили с помощью инструментов определенной среды разработки под названием Microsoft Visual Studio 2015.

Раздел II. Постановка технического задания для программной реализации модели

*Разработка и функции комплекса*

Создание диалогового технологического комплекса для генерации сложных программных систем является одним из перспективных направлений в сфере программной инженерии. Поэтому разработка и усовершенствование такого вида работы позволяет задать вектор направления в развитии проекта.

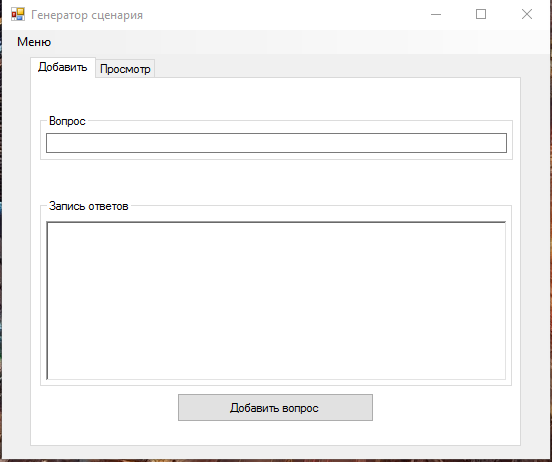
Основой для разработки данного комплекса послужила модель, использующая как основу автомат Мили. Данный курсовой проект ­– лишь макет, который может быть переработан и усовершенствован. Помимо этого, он так же содержит диалоговый технологический комплекс генерации сложных программных систем.

Программа позволяет:

* Сгенерировать диалог на основе модели, использующей принципы автомата Мили;
* Использовать уже имеющийся у пользователя сценарий диалога;
* На примере человеко-машинного взаимодействия ответить на вопросы диалога с записью отчётов по каждому из пунктов в файл;

Программа написана на языке C# в среде программирования Microsoft Visual Studio 2015 Community.

*Разработка сценария диалога*



Создание сценария, построенного на основе автомата Мили начинается с создания пустого документа в формате .xml

XML — расширяемый язык разметки.

Для записи состояния (вершины) графа используется инструмент textbox, в который записывается текст вопроса.

Переходы реализованы внутренними преобразованиями данных, а именно привязыванием входных, выходных значений и последующих вершин.

После заполнения данных необходимо нажать кнопку «Добавить вопрос». Файл формируется и обновляется автоматически.

После выполнения этих действий мы можем открыть наш созданный файл в главном окне программе. После таких манипуляций пользователю предлагается пройтись по сценарию диалога и ответить на вопросы. Исходя из ответа пользователя, система переходит в необходимое нам состояние и записывает результат в «Ход диалога».

Как говорилось выше, комплекс построен таким образом, что на его основе можно реализовать любую модель.

В ходе диалога присутствует анализатор ошибок, который выводит в поле «Сообщение об ошибке» информацию пользователю о том, что он ввёл неверный ответ и система вновь просит его ответить на вопрос, выбрав один из предложенных вариантов ответа.

В программе предусмотрен инструмент «Выпадающее меню», с помощью которого пользователь может работать в программе.

В разделе «Меню» можно перейти к работе со сценарием, выйти из системы, открыть файл диалога, сохранить ход диалога, а также найти всю сопутствующую информацию.

*Используемые инструменты для реализации*

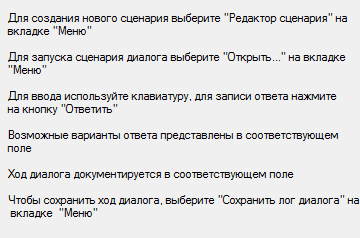
TextBox – позволяет вводить информацию в поле.



Buttons – кнопки для управления записями в таблице.



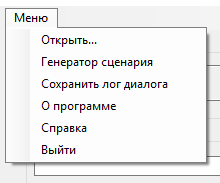
Label – надпись для именования различных объектов.



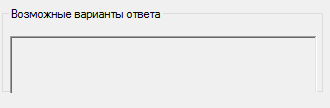
TabControl – возможность переключать вкладки.



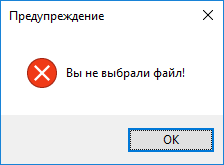
MenuStrip – выпадающее меню с различными вариантами.



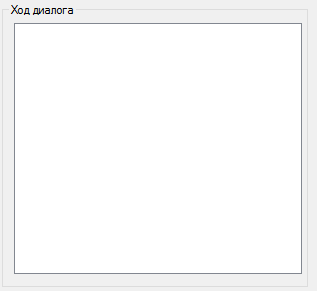
GroupBox – группирует объекты в единую систему.



MessageBox – окно с сообщением (информацией).



ListBox – используется для записи информации построчно.



Стандарты ГОСТ/ISO

Административное управление определяется:

ISO 12207:1995 – процессы жизненного цикла программных средств

ГОСТ 19.102-77 – стадии разработки ПС.

ISO 16326:1999 – руководство по применению (207 ИСО) при административном управлении проектом.

Тестирование определяется:

ANSI/IEEE 1008-1986 – тестирование программных модулей и компонентов ПС.

ANSI/IEEE 1012-1986 – планирование верификации и подтверждение достоверности качества (валидации ПС).

Документирование определяется:

ГОСТ 19105-78 (общее требование к программным документам)

ГОСТ 19106-78 (требование к программным документам, выполненным печатным способом)

ISO 9294:1990 (Руководство по управлению документированию ПО)

ISO 9127:1990.

Оценка качества:

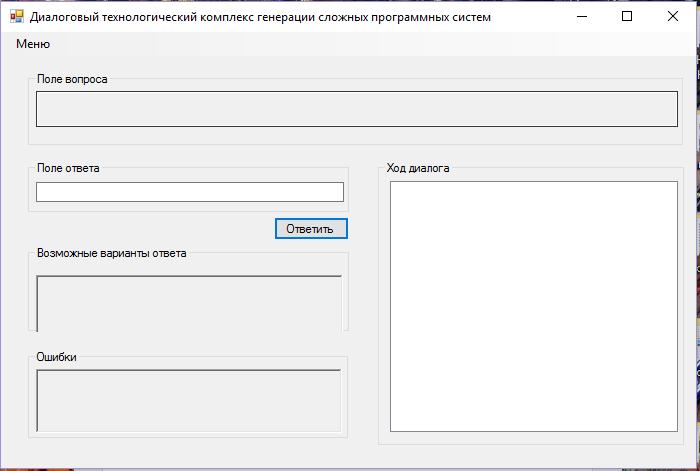
ISO 9126:1991 Оценка программного продукта характеристики качества и руководства по их применению.

ISO 14598-1-6:1998-2000 Оценивание программного продукта.

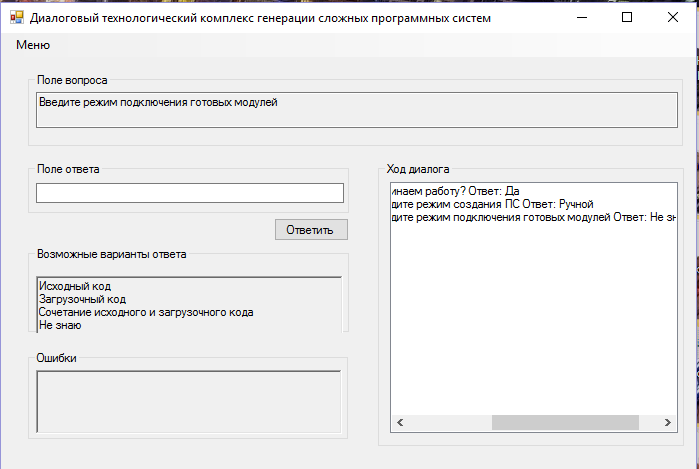
Раздел III. Представление результатов работы, полученных в ходе разработке программного обеспечения

*Примеры работы программы*

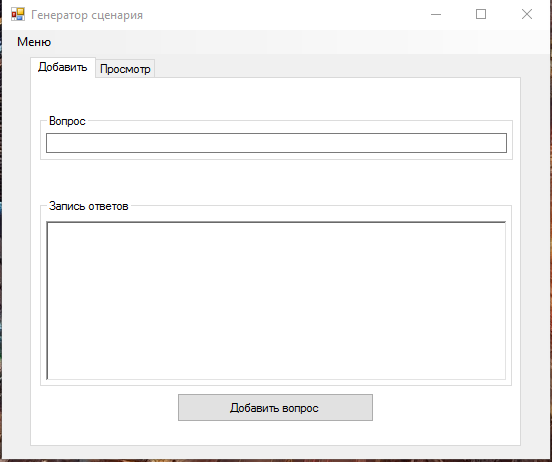
Главный экран программы:



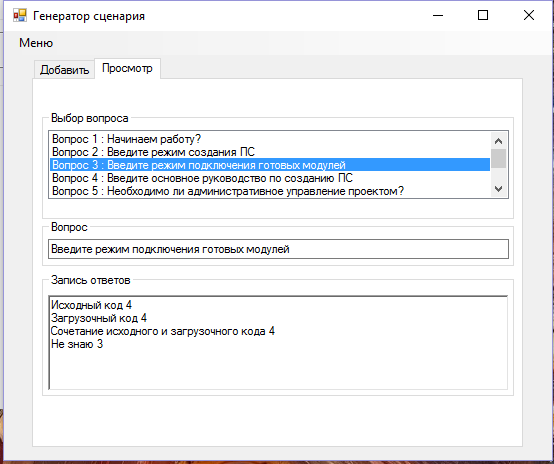
Главный экран в ходе выполнения диалога:



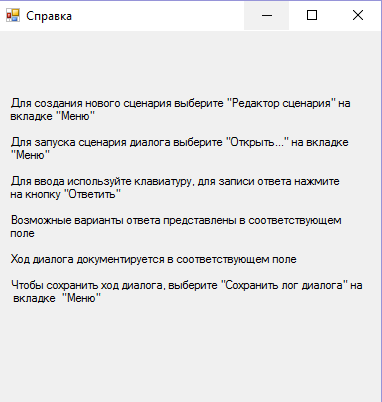
Генератор диалога:



Генератор в ходе работы:



Окно справки:



*Листинг исходного кода*

**DialogueForm.cs (main):**

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;

using System.Xml.Linq;

using System.Xml;

using System.IO;

using System.Xml.Serialization;

namespace CourseProject

{

public partial class DialogueForm : Form

{

public DialogueForm()

{

InitializeComponent();

}

string k;

private void открытьФайлToolStripMenuItem\_Click(object sender, EventArgs e)

{

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void textBox2\_TextChanged(object sender, EventArgs e)

{

}

private void DialogueForm\_Load(object sender, EventArgs e)

{

}

private void оПрограммеToolStripMenuItem\_Click(object sender, EventArgs e)

{

About f = new About();

f.Show();

}

private void справкаToolStripMenuItem\_Click(object sender, EventArgs e)

{

}

private void редакторСценарияToolStripMenuItem\_Click(object sender, EventArgs e)

{

Editor f = new Editor();

f.Show();

}

private void выйтиToolStripMenuItem\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void справкаToolStripMenuItem1\_Click(object sender, EventArgs e)

{

SprM f = new SprM();

f.Show();

}

string m = "1";

private void открытьToolStripMenuItem\_Click(object sender, EventArgs e)

{

//задаем путь к нашему рабочему файлу XML

OpenFileDialog open = new OpenFileDialog();

open.InitialDirectory = "c:\\";

open.Filter = "xml files (\*.xml)|\*.xml";

open.FilterIndex = 2;

open.RestoreDirectory = true;

if (open.ShowDialog() == DialogResult.OK)

{

k = open.FileName;

//читаем данные из файла

XDocument doc = XDocument.Load(k);

foreach (XElement el in doc.Root.Elements())

{

if (el.Attribute("id").Value == m)

{

textBox1.Text = el.Attribute("text").Value;

foreach (XElement element in el.Elements())

{

richTextBox1.Text = richTextBox1.Text + element.Value + '\n';

}

}

}

}

}

private void button1\_Click(object sender, EventArgs e)

{

int find = richTextBox1.Find(textBox1.Text);

if (find >= 0)

{

listBox1.Items.Add("Вопрос: " + textBox1.Text + " Ответ: " + textBox2.Text + '\n');

XDocument doc = XDocument.Load(k);

foreach (XElement el in doc.Root.Elements())

{

if (el.Attribute("id").Value == m)

{

textBox1.Text = el.Attribute("text").Value;

richTextBox1.Text = String.Empty;

richTextBox2.Text = String.Empty;

foreach (XElement element in el.Elements())

{

if (textBox2.Text == element.Value)

{

m = element.Attribute("nextid").Value;

}

richTextBox1.Text = richTextBox1.Text + element.Value + '\n';

}

}

if (m == "0")

{

listBox1.Items.Add("Диалог окончен. Вы можете сохранить ход диалога.");

textBox1.Text = String.Empty;

textBox2.Text = String.Empty;

richTextBox1.Text = String.Empty;

richTextBox2.Text = String.Empty;

}

}

textBox2.Text = String.Empty;

}

else

{

richTextBox2.Text = "Ошибка ввода.";

textBox2.Text = String.Empty;

}

}

private void сохранитьЛогДиалогаToolStripMenuItem\_Click(object sender, EventArgs e)

{

SaveFileDialog savelog = new SaveFileDialog();

savelog.InitialDirectory = "c:\\";

savelog.Filter = "txt files (\*.txt)|\*.txt";

savelog.FilterIndex = 2;

savelog.RestoreDirectory = true;

if (savelog.ShowDialog() == DialogResult.OK)

{

TextWriter writer = new StreamWriter(savelog.FileName);

foreach (var item in listBox1.Items)

writer.WriteLine(item.ToString());

writer.Close();

}

listBox1.Text = String.Empty;

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

}

}

**Editor.cs (generator):**

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;

using System.Xml.Linq;

using System.Xml;

namespace CourseProject

{

public partial class Editor : Form

{

public Editor()

{

InitializeComponent();

}

string k;

private void справкаToolStripMenuItem\_Click(object sender, EventArgs e)

{

SprEd f = new SprEd();

f.Show();

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

}

private void button2\_Click(object sender, EventArgs e)

{

}

private void contextMenuStrip1\_Opening(object sender, CancelEventArgs e)

{

}

private void открытьToolStripMenuItem\_Click(object sender, EventArgs e)

{

//задаем путь к нашему рабочему файлу XML

OpenFileDialog open = new OpenFileDialog();

open.InitialDirectory = "c:\\";

open.Filter = "xml files (\*.xml)|\*.xml";

open.FilterIndex = 2;

open.RestoreDirectory = true;

if (open.ShowDialog() == DialogResult.OK)

{

k = open.FileName;

//читаем данные из файла

XDocument doc = XDocument.Load(k);

}

}

private void button3\_Click(object sender, EventArgs e)

{

}

private void Editor\_Load(object sender, EventArgs e)

{

}

int l;

private void открытьToolStripMenuItem\_Click\_1(object sender, EventArgs e) //открываем файл сценария

{

//задаем путь к нашему рабочему файлу XML

OpenFileDialog open = new OpenFileDialog();

open.InitialDirectory = "c:\\";

open.Filter = "xml files (\*.xml)|\*.xml";

open.FilterIndex = 2;

open.RestoreDirectory = true;

if (open.ShowDialog() == DialogResult.OK)

{

k = open.FileName;

//читаем данные из файла

XDocument doc = XDocument.Load(k);

foreach (XElement el in doc.Root.Elements()) //заносим данные вопроса в листбокс

{

listBox1.Items.Add("Вопрос" + " " + el.Attribute("id").Value + " " + ":" + " " + el.Attribute("text").Value);

l = Int32.Parse(el.Attribute("id").Value);

}

}

}

private void listBox1\_SelectedIndexChanged(object sender, EventArgs e) //выбор вопроса для редактирования

{

string[] words = listBox1.SelectedItem.ToString().Split(' ');

int number = Int32.Parse(words[1]);

XDocument doc = XDocument.Load(k);

textBox2.Clear();

richTextBox1.Text = String.Empty;

foreach (XElement el in doc.Root.Elements())

{

if (el.Attribute("id").Value == number.ToString())

{

textBox2.Text = el.Attribute("text").Value;

foreach (XElement element in el.Elements())

{

richTextBox1.Text = richTextBox1.Text + element.Value + " " + element.Attribute("nextid").Value + "\n";

}

}

}

}

int maxID = 1;

private void button1\_Click\_1(object sender, EventArgs e)

{

XDocument doc = XDocument.Load(k);

string[] words = richTextBox2.Lines[0].Split(' ');

string[] words2 = richTextBox2.Lines[1].Split(' ');

XElement question = new XElement("question",

new XAttribute("id", maxID),

new XAttribute("text", textBox1.Text),

new XElement("answer", words[0], new XAttribute("nextid", words[1])),

new XElement("answer", words2[0], new XAttribute("nextid", words2[1])));

doc.Root.Add(question);

doc.Save(k);

maxID++;

l++;

listBox1.Items.Add("Вопрос" + " " + l + " " + ":" + " " + textBox1.Text);

}

private void сохранитьToolStripMenuItem\_Click(object sender, EventArgs e)

{

SaveFileDialog save = new SaveFileDialog();

save.InitialDirectory = "c:\\";

save.Filter = "xml files (\*.xml)|\*.xml";

save.FilterIndex = 2;

save.CreatePrompt = true;

save.OverwritePrompt = true;

save.RestoreDirectory = true;

save.AddExtension = true;

if (save.ShowDialog() == DialogResult.OK)

{

XmlTextWriter textWritter = new XmlTextWriter(save.FileName, Encoding.UTF8);

textWritter.WriteStartDocument();

textWritter.WriteStartElement("scenario");

textWritter.WriteEndElement();

textWritter.Close();

}

}

}

}

Раздел IV. Литература и источники

1) https://ru.wikipedia.org/wikiТеория\_автоматов

2) Орлов С.А., Цилькер Б.Я. Технологии разработки программного обеспечения

3) Уокер Ройс. Управление проектами по созданию программного обеспечения

4) Поспелов Д.А. Диалоговые системы в АСУ