ISSN 2500-1752

Международный журнал информационных технологий и энергоэффективности

Том 5 Номер 2(16)

2020

Международный журнал информационных технологий и энергоэффективности
Т.5 №2(16) 2020 г.

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

СОДЕРЖАНИЕ / CONTENT

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

Лебедев М.М., Раскатова М.В. Оптимизация скорости работы Android устройств с локальной базой данных // Международный журнал информационных технологий и энергоэффективности. – 2020. – Т. 5 № 2(16) с. – 3-21

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004.4

ОПТИМИЗАЦИЯ СКОРОСТИ РАБОТЫ ANDROID УСТРОЙСТВ С ЛОКАЛЬНОЙ БАЗОЙ ДАННЫХ

[1]Лебедев М.М., [2]Раскатова М.В.
ФГБОУ ВО «НИУ «МЭИ», Москва, Россия (111250, г. Москва, ул. Красноказарменная, д.14), e-mail: 1) leb.dev96@gmail.com, 2) marina@raskatova.ru

В статье описываются варианты работы устройства Android с локальными базами данных. Рассматриваются различные режимы использования и функции для работы с данными. Проводятся тесты и измерения времени выполнения операций в различных условиях. Составляются рекомендации по локальному хранению данных и написанию программного кода для работы с SQLite на платформе Android.

Ключевые слова: Android, SQLite, SQLiteOpenHelper.

OPTMIZATION OF OPERATION SPEED ANDROID DEVICES WITH LOCAL DATABASE

[1]Lebedev M.M., [2]Raskatova M.V.

NRU «MPEI», Moscow, Russia (111250, Moscow, street Krasnokazarmennaya, 14), e-mail: 1) leb.dev96@gmail.com, 2) marina@raskatova.ru

This article describes different variants of working with local database on Android platform. Various usage modes and functions for working with data are considered. Tests are conducted and measurements are taken of operations' execution timings in different conditions. Also, recommendations on local data storing and code development for working with SQLite on Android are given.

Keywords: Android, SQLite, SQLiteOpenHelper

Для локального хранения данных на устройствах Android всегда используют базу данных SQLite. Для совершения операций над данными и работой с базой обычно в качестве решения для Java кода используют библиотеку SQLiteOpenHelper [1] или же Object Relational Mapping библиотеки. Однако, последние, как правило, имеют меньшую производительность, сильно выигрывая в читабельности и наглядности кода [2]. В данной статье мы рассмотрим случай, когда приложению необходимо работать с локальной базой данных на устройстве Android и обрабатывать значительные объёмы данных. Поэтому будет рассмотрена возможность оптимизации скорости работы с базой данных SQLite в сочетании с библиотекой SQLiteOpenHelper.

SQLite базы данных используют динамическое типизирование. Каждая колонка имеет только указание, данные какого формата следует в ней хранить, но не вводит жёсткое

ограничение. Хранимые в SQLite базах данные относят к одному из пяти следующих типов: NULL, INTEGER, REAL, TEXT, BLOB. Все прочие типы приводятся базой SQLite к этим пяти [3]. Рассмотрим, как тип хранимых данных может повлиять на скорость работы с базой данных.

Создать колонку, имеющую «рекомендацию» типа NULL невозможно и не имеет смысла, поэтому данный случай не следует рассматривать.

INTEGER в зависимости от величины значения может сжиматься для экономии места на диске, но в процессе обработки механизмами SQLite и INTEGER, и REAL представляются, как 8-байтное значение. Поэтому операции над обоими типами не отличаются по времени выполнения.

Согласно документации SQLite, длина значений, хранимых в типах TEXT и BLOB - не ограничена. Это означает, что объём данных, над которыми проводится операция может каждый раз быть разным, и что теоретически это может влиять на время выполнения операции. Длина данных типа TEXT будет зависеть от используемой кодировки, количества символов и длин кодов этих символов. К примеру, при использовании UTF-8 (используется по умолчанию), каждый символ, входящий в диапазон таблицы ASCII, будет иметь вес в 1 байт. То есть, операция над группой из 8 символов английского алфавита даёт равную нагрузку в сравнении с операцией над типами INTEGER и REAL. Зависимость скорости выполнения операции от размера вставляемых данных также следует проверить.

Для работы с базой через библиотеку SQLiteOpenHelper следует создать класс-наследник, в котором будут переопределяться функции. Настройки для соединения с базой задаются в конструкторе класса. В простейшем случае, для задания настроек по умолчанию в конструктор следует передать контекст приложения, имя открываемой базы, фабрику курсоров (можно передать null) и текущую версию, с которой следует открыть базу. Разработаем необходимый класс и назовём его «DatabaseHelper». Конструктор с настройками по умолчанию будет выглядеть следующим образом:

```
private static final int DATABASE_VERSION = 1;
private static final String DATABASE_NAME = "appDB.db";
private static final String TAG = "UserLog.Database";

DatabaseHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
    Log.d(TAG, "DatabaseHelper: Constructor called");
}
```

Коллбэки создания базы и обновления её версии оставим пустыми, так как для каждого теста мы будем создавать таблицы заново, и их структура может отличаться от теста к тесту.

```
@Override
public void onCreate(SQLiteDatabase db) {
    Log.d(TAG, "onCreate: Creating app database");
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    Log.d(TAG, "onUpgrade: Upgrading app database from version " + oldVersion +
```

```
" to " + newVersion);
}
```

Тестирование будем проводить с помощью инструмента юнит тестирования для платформы андроид AndroidJUnit4.

При работе с данными могут выполняться операции выборки (SELECT), обновления данных (UPDATE), записи новых данных (INSERT), удаления данных (DELETE). Для начала рассмотрим операции вставки новых данных.

Алгоритм теста следует сделать следующим:

- очистка БД от существующих таблиц (если таковые имеются);
- создание таблицы с указанными характеристиками;
- операция над данными с замером времени;
- вывод результата.

Для того, чтобы полученные временные значения были более достоверными, тесты необходимо проводить сериями. Для чего создание таблицы и операцию над данными следует поместить в тело цикла, и выполнить некоторое количество раз, после чего вычислить среднее арифметическое полученных данных. В каждом рассматриваемом в дальнейшем случае будем проводить по 100 тестов с одинаковыми условиями, а затем вычислять и записывать среднее арифметическое полученных значений.

1 Операции вставки данных

1.1 Вставка с помощью ContentValues

Проведём тесты времени выполнения операций с настройками по умолчанию. Проведём вставку значений через функцию класса SQLiteDatabase – insert(String table, String nullColumnHack, ContentValues values). Для вставки значений через эту функцию ей необходимо подавать имя таблицы и объект ContentValues, в который перед каждым вызовом insert необходимо разместить соответствие: «имя колонки» ↔ «значение». Напишем в классе DatabaseHelper функцию, которую будем вызывать из теста:

```
long insertViaContentValues(Integer value, int count) {
    ContentValues contentValues = new ContentValues();
    SQLiteDatabase sqLiteDatabase = getWritableDatabase();
    long time = getCurrentTime();
    for (int i = 0; i < count; i++) {
        contentValues.put(KEY_COLUMN_1, value);
        sqLiteDatabase.insert(TABLE_NAME, null, contentValues);
    }
    time = getDifference(time);
    Log.d(TAG, "insertViaContentValues: Done inserting " + count + " values in "
+ time + " ms");
    return time;
}
```

Кратко запишем условия выполнения (таблица 1) и результаты теста (таблица 2).

Таблица 1 – Условия выполнения теста

Настройки SQLiteOpenHelper	**Устройство**	**Структура БД**	**Операция**
По умолчанию	Pixel 2 API 24 (AVD)	Таблица: Колонка Integer	Вставка значений через ContentValues

Таблица 2 – Результаты теста

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**
1	4,73000
10	4,50500
100	4,55500
1000	4,50835

Как можно увидеть из результатов теста – количество времени, затрачиваемого на операцию вставки при данных условиях, практически не изменяется с ростом количества идущих подряд операций. И на каждую операцию в среднем требуется около 4,6 мс.

1.2 Влияние количества колонок на операцию вставки данных

Узнаем, как количество колонок вставляемых данных влияет на скорость выполнения операций. Проведём аналогичные тесты, создав две одинаковых колонки типа Integer. Ожидаем увидеть, увеличение времени выполнения каждой операции.

Чтобы установить зависимость времени выполнения операции вставки от количества колонок, видоизменим функцию вставки для работы с переменным количеством колонок и проведём тесты с различным количеством.

```
long insertViaContentValues(Integer value, int count, int columnCount) {
    ContentValues contentValues = new ContentValues();
    SQLiteDatabase sqLiteDatabase = getWritableDatabase();
    long time = getCurrentTime();
    for (int i = 0; i < count; i++) {
        for (int j = 1; j <= columnCount; j++) {
            contentValues.put(KEY_COLUMN + j, value);
        }
        sqLiteDatabase.insert(TABLE_NAME, null, contentValues);
    }
    time = getDifference(time);
    Log.d(TAG, "insertViaContentValues: Done inserting " + count + " values in "
+ time + " ms");
    return time;
}
```

Так как SQLite базы данных имеют ограничение на количество колонок, а также параметров в выражении = 999, то вместо варианта с 1000 колонок, мы рассмотрим 999 в качестве максимума. Условия теста и результаты приведены в таблицах 3 и 4.

Таблица 3 – Условия выполнения теста

Настройки SQLiteOpenHelper	**Устройство**	**Структура БД**	**Операция**
По умолчанию	Pixel 2 API 24 (AVD)	Таблица: Колонка Integer ×X (переменное количество)	Вставка значений через ContentValues

Таблица 4 – Результаты теста

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**						
	Количество колонок:	**1**	**10**	**100**	**200**	**500**	**999**
1	4,80000		4,80000	5,42000	7,13000	14,11000	36,20000
10	4,81100		4,47200	4,99200	5,74500	7,60300	9,30300
100	4,64560		4,54880	5,01020	5,74500	7,77990	9,20260
1000	4,62806		4,50898	4,92928	5,63511	7,78070	9,26118

Глядя на полученные результаты, можно отметить, что время выполнения вставки в одну колонку не сильно меньше времени вставки в 100 колонок. С учётом того, что в мобильных приложениях, как правило, таблицы имеют небольшое количество колонок, которое не часто превышает 100, можно сказать, что увеличение числа колонок обычно почти не сказывается на производительности операций вставки данных.

Также отметим, что с ростом количества колонок уменьшается время выполнения операции, если некоторое их количество идёт подряд, вероятно, из-за внутренних механизмов оптимизации SQLite.

1.3 Влияние типов данных и длины данных

Проведём тест со строковыми данными, изменяя их длину. Ожидаем, что вставка строки длиной 8 байт эквивалентна по нагрузке вставке одного целочисленного значения, а потому время выполнения операции должно быть одинаковым, однако с ростом длины строки нагрузка должна возрастать.

Чтобы сравнить получаемые результаты с результатами теста влияния количества колонок, протестируем вставку строк объёмами эквивалентными 1, 10, 100… значениям типа Integer, как в упомянутом тесте. Условия теста и результаты приведены в таблицах 5 и 6.

Таблица 5 – Условия выполнения теста

Настройки SQLiteOpenHelper	**Устройство**	**Структура БД**	**Операция**
По умолчанию	Pixel 2 API 24 (AVD)	Таблица: Колонка String	Вставка значений через ContentValues

Таблица 6 – Результаты теста

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**						
	Длина строки в байтах:	**1*8**	**10*8**	**100*8**	**200*8**	**500*8**	**999*8**
1	4,89000		4,94000	5,13000	5,22000	5,11000	7,34000
10	4,52700		4,75800	5,08000	5,50000	6,11100	6,80700
100	4,53240		4,79780	5,19250	5,59230	6,29870	6,74740
1000	4,52082		4,67494	5,19367	5,52837	6,25584	6,63281

Получаемые результаты говорят нам о том, что увеличение объёма вставляемых за одну операцию данных, как и ожидалось, приводит к увеличению времени выполнения операции. Однако, куда выгоднее по времени вставить строку эквивалентную по объёму 100 целочисленным значениям, чем разместить в 100 колонок целочисленные значения. Наблюдаемое явление заставляет задуматься об использовании сериализации данных, например, в JSON строку. Если количество колонок действительно велико, это может положительно сказаться на производительности.

1.4 Объединение в одну транзакцию

Проведём тестирование времени выполнения операций вставки, используя механизм транзакций SQLite. Перепишем функцию так, чтобы каждый цикл операций был обёрнут в цельную транзакцию, и посмотрим на результаты (таблицы 7, 8).

```
long insertViaContentValuesWithTransaction(Integer value, int count, int
columnCount) {
    ContentValues contentValues = new ContentValues();
    SQLiteDatabase sqLiteDatabase = getWritableDatabase();
    long time = getCurrentTime();
    sqLiteDatabase.beginTransaction();
    try {
        for (int i = 0; i < count; i++) {
            for (int j = 1; j <= columnCount; j++) {
                contentValues.put(KEY_COLUMN + j, value);
            }
            sqLiteDatabase.insert(TABLE_NAME, null, contentValues);
        }
        sqLiteDatabase.setTransactionSuccessful();
```

```
    } finally {
        sqLiteDatabase.endTransaction();
    }
    time = getDifference(time);
    Log.d(TAG, "insertViaContentValuesWithTransaction: Done inserting " + count
+ " values in " + time + " ms");
    return time;
}
```

Таблица 7 – Условия выполнения теста

Настройки SQLiteOpenHelper	Устройство	Структура БД	Операция
По умолчанию	Pixel 2 API 24 (AVD)	Таблица: Колонка Integer	Вставка значений через ContentValues в единой транзакции

Таблица 8 – Результаты теста

Количество операций подряд	Среднее время на выполнение одной операции (мс)
1	5,210000
10	0,697000
100	0,199300
1000	0,147860
10000	0,141041

Время выполнения операций резко сократилось. Особенно это заметно по результатам вставки большого количества строк, причём чем больше вставок, тем быстрее выполняется каждая из них. Становится очевидно, что вставку нескольких строк по возможности всегда лучше объединять в единую транзакцию, так как, например, при проведении тестов вставка 100 строк без транзакции заняла примерно 450 мс, а в единой транзакции – всего 20 мс.

1.5 ExecSQL

Протестируем иной способ вставки данных – через выполнение функции ExecSQL. Данный метод компилирует и выполняет любой запрос к БД, кроме SELECT.

```
long insertViaExecSQL(String value, int count) {
    SQLiteDatabase sqLiteDatabase = getWritableDatabase();
    long time = getCurrentTime();
    String query = "Insert into " + TABLE_NAME + " (" + KEY_COLUMN_1 + ") values
(?)";
    for (int i = 0; i < count; i++) {
        sqLiteDatabase.execSQL(query, new String[] {value});
    }
    time = getDifference(time);
    Log.d(TAG, "insertViaExecSQL: Done inserting " + count + " values in " +
time + " ms");
```

```
    return time;
}
```

Условия теста и результаты приведены в таблицах 9 и 10.

Таблица 9 – Условия выполнения теста

Настройки SQLiteOpenHelper	**Устройство**	**Структура БД**	**Операция**
По умолчанию	Pixel 2 API 24 (AVD)	Таблица: Колонка Integer	Вставка значений через ExecSQL

Таблица 10 – Результаты теста

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**
1	4,91000
10	4,60600
100	4,61530
1000	4,63406

Очевидно, что данный способ оказался малоэффективен, если не менее эффективен, чем вставка через ContentValues.

1.6 Prepared Statements

Протестируем вставку данных, используя механизм прекомпилированных запросов. Такие запросы компилируются единожды, а в последствии могут быть переиспользованы базой некоторое количество раз подряд. Их рекомендуется использовать, если вы собираетесь выполнить некоторую повторяющуюся операцию, например, вставку нескольких строк. Прекомпилированные запросы являются предпочтительными при использовании повторяющихся операций, к тому же их использование сокращает количество используемой памяти [4].

```
long insertViaPrecompiledStatement(Integer value, int count) {
    SQLiteDatabase sqLiteDatabase = getWritableDatabase();
    long time = getCurrentTime();
    String query = "Insert into " + TABLE_NAME + " (" + KEY_COLUMN_1 + ") values
(?)";
    SQLiteStatement sqLiteStatement = sqLiteDatabase.compileStatement(query);
    for (int i = 0; i < count; i++) {
        sqLiteStatement.bindLong(1, value);
        sqLiteStatement.executeInsert();
    }
    time = getDifference(time);
    Log.d(TAG, "insertViaPrecompiledStatement: Done inserting " + count + "
values in " + time + " ms");
    return time;
}
```

Условия теста и результаты приведены в таблицах 11 и 12.

Таблица 11 – Условия выполнения теста

Настройки SQLiteOpenHelper	Устройство	Структура БД	Операция
По умолчанию	Pixel 2 API 24 (AVD)	Таблица: Колонка Integer	Вставка значений через PrecompiledStatement

Таблица 12 – Результаты теста

Количество операций подряд	Среднее время на выполнение одной операции (мс)
1	4,89000
10	4,43400
100	4,45830
1000	4,45320

В результатах теста заметно небольшое сокращение времени выполнения, по сравнению с обычной вставкой через ContentValues.

1.7 Журналирование

Попробуем изменить настройки открытия БД, обернув конструктор DatabaseHelper в фабричный статический метод.

```
@RequiresApi(api = Build.VERSION_CODES.P)
static DatabaseHelper databaseHelperFactoryMethod(Context context) {
    SQLiteDatabase.OpenParams.Builder builder = new
SQLiteDatabase.OpenParams.Builder();
    builder.setJournalMode("TRUNCATE");
    return new DatabaseHelper(context, builder.build());
}

@RequiresApi(api = Build.VERSION_CODES.P)
DatabaseHelper(Context context, SQLiteDatabase.OpenParams openParams) {
    super(context, DATABASE_NAME, DATABASE_VERSION, openParams);
    Log.d(TAG, "DatabaseHelper: Constructor called");
}
```

К сожалению, вызов такого конструктора требует минимального уровня API = 28, что заставит нас поднять минимальный уровень API для использования приложения, и, возможно, ощутимо сократить круг мобильных устройств, которым оно будет доступно. Поэтому рассматривать возможность изменения параметров открытия базы следует исключительно в случаях, когда известно, что приложение будет использоваться более современными моделями устройств.

Изменим параметр журналирования. Журналирование БД может работать в одном из нескольких доступных режимов.

- DELETE – Стандартное поведение, при котором журнал отката удаляется по завершению транзакции.
- TRUNCATE – По завершению транзакции файл журнала обрезается до нулевой длины.
- PERSIST – Файл журнала остаётся в системе, но его заголовок заполняется нулями.
- MEMORY – Заставляет поместить файл отката в оперативную память, что повышает скорость, но может привести к повреждению БД.
- WAL – Использует Write Ahead Logging. Полезно при использовании БД одновременно несколькими процессами.
- OFF – Отключает журнал отката, из-за чего функция ROLLBACK будет вести себя непредсказуемо, не рекомендуется к использованию.

Не следует использовать режимы MEMORY и OFF в реальных условиях эксплуатации приложения, так как это ставит под угрозу целостность данных в базе, а потому их рассмотрение также можно опустить. Условия теста и результаты приведены в таблицах 13 и 14.

Таблица 13 – Условия выполнения теста

Настройки SQLiteOpenHelper	**Устройство**	**Структура БД**	**Операция**
Различные режимы журналирования	Pixel 2 API 28 (AVD)	Таблица: Колонка Integer	Вставка значений через ContentValues

Таблица 14 – Результаты теста

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**				
	Используемый режим:	**DELETE**	**TRUNCATE**	**PERSIST**	**WAL**
1	6,24000		8,56000	7,45000	1,62000
10	5,68200		8,07700	6,93200	1,22100
100	5,61160		7,99180	6,86410	1,28850
1000	5,51070		7,82960	6,81806	1,27050

Результаты тестов показывают, что наиболее эффективно использовать режим WAL для осуществления вставки записей. Аналогичные результаты были получены при выполнении тестов вставки через ContentValues с использованием транзакций. Условия теста и результаты приведены в таблицах 15 и 16.

Таблица 15 – Результаты теста вставок в единой транзакции

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**				
	Используемый режим:	**DELETE**	**TRUNCATE**	**PERSIST**	**WAL**
1	6,88000		8,99000	7,11000	1,97000
10	0,74100		0,95700	0,79200	0,27100
100	0,17840		0,20000	0,18660	0,13200
1000	0,12178		0,12275	0,12114	0,11309

1.8 Write Ahead Logging

При тестировании различных режимов журналирования было выявлено сокращение времени выполнения операций вставки данных в несколько раз при использовании режима WAL (Write Ahead Logging). Открытие базы с нестандартными опциями доступно только на более новых версиях платформы Android, что доставляет неудобства при разработке приложений для широкого круга пользователей. Однако режим WAL можно включить или отключить в ходе выполнения кода программы в любой момент времени. Согласно официальной документации SQLite [5], данный режим имеет следующие наиболее значимые особенности:

- WAL значительно быстрее в большинстве сценариев работы БД
- WAL повышает эффективность параллельного использования БД, потому как читатели не блокируют писателей, а также верно и обратное
- Величина страницы не может быть изменена в этом режиме
- WAL может быть на 1-2% менее эффективен при частых чтениях и редких записях

Проведём тестирование вставки через ContentValues без транзакции и с транзакцией. Для отслеживания влияния режима попробуем провести тест без него (в режиме по умолчанию), затем включим его в начале операции и выключим в конце, а затем установим его до выполнения тестов. Условия теста и результаты приведены в таблицах 16 - 18.

Таблица 16 – Условия выполнения теста

Настройки SQLiteOpenHelper	**Устройство**	**Структура БД**	**Операция**
По умолчанию, включение WAL «на ходу», постоянно включённый WAL	Pixel 2 API 24 (AVD)	Таблица: Колонка Integer	Вставка значений через ContentValues с транзакцией и без

Таблица 17 – Результаты теста без транзакции

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**			
	Используемый режим:	**По умолчанию**	**WAL на время выполнения операции**	**Постоянно включённый WAL**
1	4,98000		34,0300	1,31000
10	4,62500		4,31000	0,99100
100	4,60180		2,60100	0,98470
1000	4,52814		1,16059	0,95566

Таблица 18 – Результаты теста в единой транзакции

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**			
	Используемый режим:	**По умолчанию**	**WAL на время выполнения операции**	**Постоянно включённый WAL**
1	5,10000		34,3600	1,66000
10	0,64900		2,54100	0,25300
100	0,19610		0,36960	0,15340
1000	0,14653		0,16629	0,13960

На основе произведённых тестов, можно заметить, что включение и отключение данного режима требует некоторого времени, поэтому его не стоит включать или отключать во время выполнения операций. Очевидно также, что использование режима на постоянной основе действительно даёт уменьшение времени выполнения операций вставки. При написании кода следует учесть, какие сценарии работы с базой данных будут использоваться наиболее часто. Так как, согласно документации, время чтения может быть увеличено на несколько процентов, а, согласно проведённым тестам, время выполнения операций вставки может сократиться до нескольких раз, режим WAL можно смело рекомендовать к включению в приложениях, чаще (или в равной степени часто) использующих операции вставки, чем операции получения данных.

2 Операции чтения

2.1 Чтение с помощью Query

Рассмотрим операции чтения из БД. Функция Query библиотеки SQLite – одна из функций, дающих возможность прочитать данные из базы. По вызову функции из БД выбираются данные, подходящие по параметрам поиска, а в код программы передаётся курсор для навигации по результатам. Перед выполнением операций чтения необходимо создать таблицу и заполнить её данными. Проделаем это перед выполнением тестов: создадим таблицу с одной целочисленной колонкой и вставим туда 1000 записей, после чего будем выполнять уже сам тест чтения. Условия теста и результаты приведены в таблицах 19 и 20.

Таблица 19 – Условия выполнения теста

Настройки SQLiteOpenHelper	**Устройство**	**Структура БД**	**Операция**
По умолчанию	Pixel 2 API 24 (AVD)	Таблица: Колонка Integer 1000 записей	Чтение значений с помощью функции Query

Таблица 20 – Результаты теста без транзакции

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**
1	1,03000
10	0,10300
100	0,01220
1000	0,00288

По результатам теста можно заметить, что чем больше строк считывается подряд, тем меньше времени занимает каждая операция. Это объясняется тем, что на выполнение получения результата тратится всегда несколько больше времени, чем занимает непосредственно каждая операция считывания данных курсором.

2.2 Влияние количества записей в таблице

Увеличим число записей в таблице и сравним результаты, при этом считывать мы будем также только некоторое количество записей сверху. Условия теста и результаты приведены в таблицах 21 и 22.

Таблица 21 – Условия выполнения теста

Настройки SQLiteOpenHelper	**Устройство**	**Структура БД**	**Операция**
По умолчанию	Pixel 2 API 24 (AVD)	Таблица: Колонка Integer Переменное количество записей	Чтение значений с помощью функции Query

Таблица 22 – Результаты теста

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**				
	Количество записей в таблице:	**1000**	**10000**	**100000**	**1000000**
1	1,10000		14,7200	816,370	1666,69
10	0,10800		1,46500	81,7810	166,476
100	0,01280		0,14690	8,17270	16,6423
1000	0,00284		0,01628	0,81900	1,66530
10000	0,00029		0,00358	0,08368	0,16830

По результатам теста можно отметить, что среднее время выполнения одной операции уменьшается примерно во столько же раз, во сколько и увеличивается количество операций. Это означает, что увеличение количества считываемых данных почти не влияет на время выполнения одной операции чтения, а вот увеличение записей в таблице и, как следствие, количества результатов поиска резко негативно сказывается на времени чтения. То есть не столь важно, читаете ли вы одну запись из огромной таблицы или 100, чтение займёт примерно одинаковое количество времени.

2.3 Влияние количества колонок

Установим количество записей – 10000. Увеличим количество колонок и видоизменим функцию получения записей, чтобы она могла работать с переменным количеством колонок. Условия теста и результаты приведены в таблицах 23 и 24.

Таблица 23 – Условия выполнения теста

Настройки SQLiteOpenHelper	**Устройство**	**Структура БД**	**Операция**
По умолчанию	Pixel 2 API 24 (AVD)	Таблица: Переменное количество колонок Integer 10000 записей	Чтение значений с помощью функции Query

Таблица 24 – Результаты теста

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**					
	Количество колонок:	**1**	**10**	**100**	**200**	**500**
1	14,2500		70,6900	144,520	255,560	599,32001
10	1,41100		7,02800	14,5220	25,2710	59,97800
100	0,14120		0,71460	1,53440	2,68870	6,47050
1000	0,01578		0,08110	0,24959	0,52306	1,39602
10000	0,00347		0,02124	0,21253	0,49730	2,01422

2.4 Влияние объёма данных

Проведём тест аналогичный тестам записи. Будем считывать строки различной длины, постепенно увеличивая объём, соблюдая его эквивалентность объёму некоторого количества целочисленных значений. Будем сравнивать эффективность выполнения операций над длинными строками с операциями над большим количеством целочисленных значений. Условия теста и результаты приведены в таблицах 25 и 26.

Таблица 25 – Условия выполнения теста

Настройки SQLiteOpenHelper	**Устройство**	**Структура БД**	**Операция**
По умолчанию	Pixel 2 API 24 (AVD)	Таблица: Колонка String 10000 записей	Чтение значений с помощью функции Query

Таблица 26 – Результаты теста

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**					
	Длина строки в байтах:	**1*8**	**10*8**	**100*8**	**200*8**	**500*8**
1	15,3700		16,6000	13,80000	19,73000	32,48000
10	1,52900		1,65200	1,37400	1,99700	3,31000
100	0,15530		0,16840	0,15510	0,23360	0,41400
1000	0,01833		0,02124	0,03339	0,05790	0,13778
10000	0,00501		0,00713	0,02688	0,05431	0,14571

Если сравнить результаты теста с результатами чтения из множества колонок, то можно отметить значительное уменьшение времени на выполнение операций. Это ещё один плюс в пользу сериализации данных в единую строку, так как и операции чтения, и операции вставки с одной длинной строкой куда выгоднее операций над множеством колонок.

2.5 Объединение в единую транзакцию

Объединим операции чтения данных в одну транзакцию, аналогично, как было сделано в операциях вставки в единой транзакции. Условия теста и результаты приведены в таблицах 27 и 28.

Таблица 27 – Условия выполнения теста

Настройки SQLiteOpenHelper	**Устройство**	**Структура БД**	**Операция**
По умолчанию	Pixel 2 API 24 (AVD)	Таблица: Колонка Integer Переменное количество записей	Чтение значений с помощью функции Query в единой транзакции

Таблица 28 – Результаты теста

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**				
	Количество записей в таблице:	1000	10000	100000	1000000
1	1,47000		14,5400	824,760	1707,85
10	0,14000		1,43600	82,4720	171,430
100	0,01340		0,14540	8,26910	17,0568
1000	0,00312		0,01621	0,82716	1,70652
10000	0,00031		0,00350	0,08453	0,17345

Время выполнения операций осталось практически неизменным. Поэтому данный подход не имеет смысла применять на операциях чтения.

2.6 RawQuery

Существует альтернативная функция для получения данных из БД, которая также возвращает курсор для чтения и страницу результатов. Это функция RawQuery. Она принимает на вход «сырой» SQL запрос, который компилирует, подставляя переданные параметры, если таковые имеются, и затем выполняет. Условия теста и результаты приведены в таблицах 29 и 30.

Таблица 29 – Условия выполнения теста

Настройки SQLiteOpenHelper	**Устройство**	**Структура БД**	**Операция**
По умолчанию	Pixel 2 API 24 (AVD)	Таблица: Колонка Integer Переменное количество записей	Чтение значений с помощью функции RawQuery

Таблица 30 – Результаты теста

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**				
	Количество записей в таблице:	**1000**	**10000**	**100000**	**1000000**
1	1,22000		14,3400	838,299	1717,10
10	0,12700		1,42300	83,4710	172,027
100	0,01600		0,14580	8,39010	17,2879
1000	0,00577		0,01829	0,84256	1,72294
10000	0,00059		0,00599	0,08863	0,17584

В целом, можно утверждать, что использование данной функции не оказало значимого положительного эффекта на время выполнения операций. В основном, результаты оказались даже немного хуже, чем при использовании обычной функции Query.

2.7 Write Ahead Logging

Особенности данного режима уже были рассмотрены ранее при рассмотрении операций вставки записей. Следует включить режим и проверить, не падает ли эффективность выполнения операций считывания более, чем это заявлено в документации (1-2%). Условия теста и результаты приведены в таблицах 31 и 32.

Таблица 31 – Условия выполнения теста

Настройки SQLiteOpenHelper	**Устройство**	**Структура БД**	**Операция**
Включённый режим WAL	Pixel 2 API 24 (AVD)	Таблица: Колонка Integer Переменное количество записей	Чтение значений с помощью функции Query

Таблица 32 – Результаты теста

Количество операций подряд	**Среднее время на выполнение одной операции (мс)**				
	Количество записей в таблице:	**1000**	**10000**	**100000**	**1000000**
1	1,17000		14,5000	829,690	1710,03
10	0,11300		1,45100	82,9590	171,332
100	0,01630		0,14710	8,30550	17,0311
1000	0,00571		0,01851	0,83657	1,71319
10000	0,00057		0,00607	0,08829	0,17441

Очевидно, что время выполнения операций чтения действительно практически не изменилось, как и было указано в документации.

Операции вставки строк следует оборачивать в транзакции. Это сильно ускоряет процесс выполнения операции, особенно если требуется большое количество вставок.

Чтобы ещё более сократить время выполнения операции вставок следует включить режим WAL на постоянной основе, однако следует предварительно рассмотреть наиболее часто используемые операции при работе с БД, так как наибольшая выгода использования этого режима – при частых вставках в базу.

Для вставки данных рекомендуется использовать функцию insert библиотеки SQLiteDatabase или прекомпилированные запросы (последние наиболее предпочтительны). Для чтения следует использовать функции query и rawQuery.

Большое количество колонок показало негативное влияние как на время выполнения вставки данных, так и на выполнение операций чтения данных. В случаях с большим количеством колонок следует рассмотреть возможность сериализации данных и их упаковки в одну.

Список литературы

1. SQLiteOpenHelper | Для разработчиков Android | Android Developers / Android developers – Режим доступа: https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper, свободный. (Дата обращения: 11.04.2020 г.).
2. Марашан М. В. Сравнение производительности ORM-библиотек как критерия выбора для работы с базой данных SQLite на устройствах с ОС Android // Молодой учёный. — 2016. — №26. — С. 146-149. — Режим доступа: https://moluch.ru/archive/130/36100/, свободный. (Дата обращения: 22.03.2020 г.).
3. Sunny K. A, Vikash K. K., Android SQLite Essentials. United Kingdom: Packt 2014. 110p.
4. SQLite Performance and Prepared Statements -- Visual Studio Magazine [Электронный ресурс] / Visual Studio Magazine – Режим доступа:

https://visualstudiomagazine.com/articles/2014/03/01/sqlite-performance-and-prepared-statements.aspx, свободный. (Дата обращения: 10.04.2020 г.).
5. Write-Ahead Logging [Электронный ресурс] / SQLite Org – Режим доступа: https://www.sqlite.org/wal.html, свободный. (Дата обращения: 04.04.2020 г.).

References

1. SQLiteOpenHelper | For Android Developers | Android Developers / Android developers – Access mode: https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper, free. (Request date: 11.04.2020 y.).
2. Marshan M. V. ORM libraries performance comparison as a selection criterion for working with the SQLite database on Android devices // Young scientist. — 2016. — №26. — P. 146-149. — Access mode: https://moluch.ru/archive/130/36100/, free. (Request date: 22.03.2020 y.).
3. Sunny K. A, Vikash K. K., Android SQLite Essentials. United Kingdom: Packt 2014. 110p.
4. SQLite Performance and Prepared Statements -- Visual Studio Magazine [Electronic resource] / Visual Studio Magazine – Access mode: https://visualstudiomagazine.com/articles/2014/03/01/sqlite-performance-and-prepared-statements.aspx, free. (Request date: 10.04.2020 y.).
5. Write-Ahead Logging [Electronic resource] / SQLite Org – Access mode: https://www.sqlite.org/wal.html, free. (Request date: 04.04.2020 y.).
1. The use of corrective capabilities of codes to ensure fault tolerance. Petlevanny S.V., Sagdeev A.K. Modern high technology. 2007. No.4. P. 41-42

Сидоров И.А. Разработка композиции нечетких автоматов и способа выбора сценариев управления рисками на ее основе // Международный журнал информационных технологий и энергоэффективности. – 2020. – Т. 5 № 2(16) с. 22–35

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004.42

РАЗРАБОТКА КОМПОЗИЦИИ НЕЧЕТКИХ АВТОМАТОВ И СПОСОБА ВЫБОРА СЦЕНАРИЕВ УПРАВЛЕНИЯ РИСКАМИ НА ЕЕ ОСНОВЕ

Сидоров И.А.
Филиал ФГБОУ ВО «Национальный исследовательский университет «МЭИ» в г. Смоленске, Россия, (214013, г. Смоленск, Энергетический проезд, 1), e-mail: igorsidorov770@gmail.com.

Статья посвящена разработке композиции нечетких автоматов и способа выбора сценариев управления рисками на ее основе. Рассмотрена терминология композиции нечетких автоматов, нечеткий автомат, композиция нечетких автоматов, способ построения композиции нечетких автоматов, сценарий управления рисками и способ выбора сценариев управления рисками.

Ключевые слова: события, риск-события, логические операторы, ситуации, риск-ситуации, мероприятия, сценарий управления рисками, композиция нечетких автоматов.

DEVELOPMENT OF A COMPOSITION OF FUZZY AUTOMATIC MACHINES AND A METHOD FOR SELECTING RISK MANAGEMENT SCENARIOS ON ITS BASIS

Sidorov I.A.
Smolensk Branch of the National Research University "Moscow Power Engineering Institute", Smolensk, Russia (214013, Smolensk, Energetichesky proezd, 1), e-mail: igorsidorov770@gmail.com.

The article is devoted to the development of a composition of fuzzy automata and a method for selecting risk management scenarios based on it. The terminology of a composition of fuzzy automata, a fuzzy automaton, a composition of fuzzy automata, a method for constructing a composition of fuzzy automata, a risk management scenario, and a method for selecting risk management scenarios are considered.

Keywords: events, risk events, logical operators, situations, risk situations, events, risk management scenario, composition of fuzzy automata.

Введение

При возрастании масштабов сложных организационно-технических систем и происходящих в них процессов все большее значение приобретает управление рисками. Особенно актуальной является разработка способа выбора сценариев управления рисками, определяющих последовательность действий, необходимых для перевода системы из одного состояния в другое, при котором общий ущерб от наступления рисков, а также расходы на мероприятия будут минимальными. Таким способом является способ выбора сценариев управления рисками на основе композиции нечетких автоматов.

Терминология композиции нечетких автоматов

Перед тем как переходить к композиции нечетких автоматов необходимо познакомиться с тем, что такое событие, риск-событие, логический оператор И, логический оператор ИЛИ, ситуация и риск-ситуация.

Событие – это то, что может произойти с определенной вероятностью [1-4]. Событие можно описать с помощью следующей формулы:

$$E_i = \langle D, P \rangle, \quad (1.1)$$

где E – событие; i – номер события; D – описание события; P – вероятность наступления события.

Риск-событие – это то, что может произойти с определенной вероятностью и нанести ущерб [1-4]. Риск-событие можно описать с помощью следующей формулы:

$$RE_i = \langle D, P, C \rangle, \quad (1.2)$$

где RE – риск-событие; i – номер риск-события; D – описание риск-события; P – вероятность наступления риск-события; C – ущерб от наступления риск-события.

Логический оператор И – данный оператор связывает входные события или риск-события с выходным событием или риск-событием. При наступление всех входных событий или риск-событий происходит наступление выходного события или риск-события [5]. Логический оператор И можно описать с помощью следующей формулы:

$$AND_i = \langle X_1, X_2, Y \rangle, \quad (1.3)$$

где AND – логический оператор И; i – номер логического оператора И; X_1 – первое входное событие или риск-событие; X_2 – второе входное событие или риск-событие; Y – выходное событие или риск-событие.

Причем вероятность выходного события или риск-события вычисляется по следующей формуле [6]:

$$P_y = \min(P_{x1}, P_{x2}), \quad (1.4)$$

где P_y – вероятность выходного события или риск-события; P_{x1} – вероятность первого входного события или риск-события; P_{x2} – вероятность второго входного события или риск-события.

На рисунке 1 представлен пример использования логического оператора И.

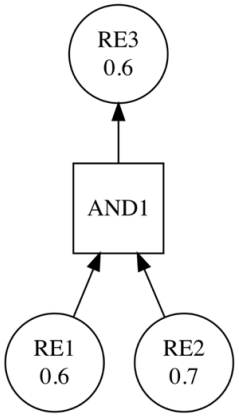

Рисунок 1 – Пример использования логического оператора И

Логический оператор ИЛИ – данный оператор связывает входные события или риск-события с выходным событием или риск-событие. При наступлении хотя бы одного входного

события или риск-события или всех событий или риск-событий происходит наступление выходного события или риск-события [5]. Логический оператор ИЛИ можно описать с помощью следующей формулы:

$$OR_i = \langle X_1, X_2, Y \rangle, \quad (1.5)$$

где OR – логический оператор ИЛИ; i – номер логического оператора ИЛИ; X_1 – первое входное событие или риск-событие; X_2 – второе входное событие или риск-событие; Y – выходное событие или риск-событие.

Причём вероятность выходного события или риск-события вычисляется по следующей формуле [6]:

$$P_y = \max(P_{x1}, P_{x2}), \quad (1.6)$$

где P_y – вероятность выходного события или риск-события; P_{x1} – вероятность первого входного события или риск-события; P_{x2} – вероятность второго входного события или риск-события.

На рисунке 2 представлен пример использования логического оператора ИЛИ.

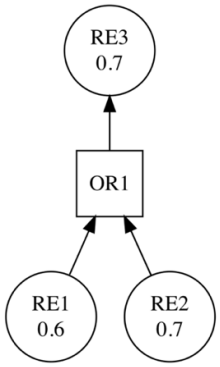

Рисунок 2 – Пример использования логического оператора ИЛИ

Иногда несколько логических операторов И или ИЛИ приводят к наступлению одного события или риск-события. Пример представлен на рисунке 3.

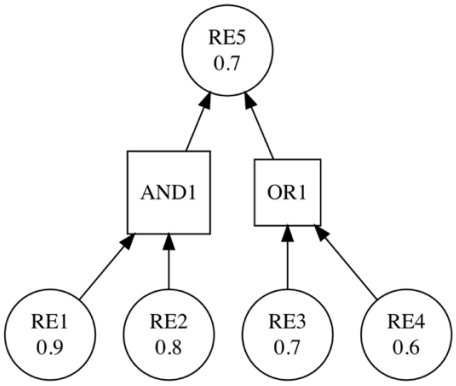

Рисунок 3 – Пример, в котором несколько логических операторов И или ИЛИ приводят к наступлению одного события или риск-события

Если несколько логических операторов И или ИЛИ приводят к наступлению одного события или риск-события, то вероятность выходного события или риск-события вычисляется по следующей формуле [6]:

$$P_y = \min(P_1, P_2, \dots, P_i, \dots, P_n), \text{при } i = 1 \dots n, \quad (1.7)$$

где P_y – вероятность выходного события или риск-события; i – номер логического оператора И или ИЛИ; P_i – вероятность выходного события или риск-события логического оператора И или ИЛИ под номером i.

Ситуация – это множество событий и риск-событий, приводящих к результирующему событию. Ситуацию можно описать с помощью следующей формулы:

$$S_{E_i} = \{e, re | e \in E, re \in RE\}, \quad (1.8)$$

где E_i – результирующие событие; S_{E_i} – ситуация, описывающая E_i; e – событие; re – риск-событие; E – множество событий; RE – множество риск-событий.

Риск-ситуация – это множество событий и риск-событий, приводящих к результирующему риск-событию. Риск-ситуацию можно описать с помощью следующей формулы:

$$RS_{RE_i} = \{e, re | e \in E, re \in RE\}, \quad (1.9)$$

где RE_i – результирующие риск-событие; S_{RE_i} – ситуация, описывающая RE_i; e – событие; re – риск-событие; E – множество событий; RE – множество риск-событий.

2. Нечеткий автомат

Композиция нечетких автоматов представляет собой множество нечетких автоматов. Поэтому необходимо для начала разобраться тем, что такое нечеткий автомат.

Структурно нечеткий автомат представляется собой ориентированный граф, узлами которого являются его состояния, а дуги характеризуют переходы между состояниями [6-8]. Нечеткий автомат можно описать с помощью следующей формулы:

$$FA = \langle Q, s, f, A, T \rangle, \quad (2.1)$$

где FA – нечеткий автомат; Q – множество состояний; s – вектор начальных состояний; f – вектор конечных состояний; A – множество входных символов (алфавит); T – множество нечетких матриц переходов.

Множество состояний можно описать с помощью следующей формулы:

$$Q = \{Q_1, Q_2, \dots, Q_i, \dots Q_n\}, \text{при } i = 1 \dots n, \quad (2.2)$$

где Q – множество состояний; i – номер состояния; Q_i – состояние под номером i.

Вектор начальных состояний можно описать с помощью следующей формулы:

$$s = (s_{Q_1}, s_{Q_2}, \dots, s_{Q_i}, \dots, s_{Q_n}), \quad (2.3)$$

где s – вектор начальных состояний; s_{Q_i} – статус состояния Q_i, если состояние начальное, то $s_{Q_i} = 1$, иначе $s_{Q_i} = 0$.

Вектор конечных состояний можно описать с помощью следующей формулы:

$$f = (f_{Q_1}, f_{Q_2}, \dots, f_{Q_i}, \dots, f_{Q_n}), \quad (2.4)$$

где f – вектор конечных состояний; f_{Q_i} – статус состояния Q_i, если состояние конечное, то $f_{Q_i} = 1$, иначе $f_{Q_i} = 0$.

Множество входных символов (алфавит) можно описать с помощью следующей формулы:

$$A = \{A_1, A_2, \dots, A_i, \dots A_m\}, \text{при } i = 1 \dots m, \quad (2.5)$$

где A – множество входных символов; i – номер входного символа; A_i – входной символ под номером i.

Множество нечетких матриц переходов можно описать с помощью следующей формулы:

$$T = \{T_{A_1}, T_{A_2}, \dots, T_{A_i}, \dots, T_{A_m}\}, \quad (2.6)$$

где T – множество нечетких матриц переходов; T_{A_i} – нечеткая матрица переходов для входного символа A_i.

Нечеткую матрицу переходов можно описать с помощью следующей формулы:

$$T_{A_i} = \begin{pmatrix} \mu_{Q_1Q_1} & \cdots & \mu_{Q_1Q_n} \\ \vdots & \ddots & \vdots \\ \mu_{Q_nQ_1} & \cdots & \mu_{Q_nQ_n} \end{pmatrix}, \quad (2.7)$$

где T_{A_i} – нечеткая матрица переходов для входного символа A_i; $\mu_{Q_jQ_k}$ – степень принадлежности перехода из состояния Q_j в состояние Q_k под действием входного символа A_i.

На нечеткий автомат можно подать входное слово, которое можно описать с помощью следующей формулы:

$$W = \{A_i | A_i \in A\}, \quad (2.8)$$

где A – множество входных символов; W – слово, состоящие из элементов множества A.

Отклик нечеткого автомата на входное слово можно определить рассчитав степень принадлежности по следующей формуле:

$$\mu_W = s \circ T_W \circ f, \quad (2.9)$$

где μ_W – степень принадлежности под воздействием входного слова W; s – вектор начальных состояний; f – вектор конечных состояний.

На рисунке 4 представлен пример нечеткого автомата.

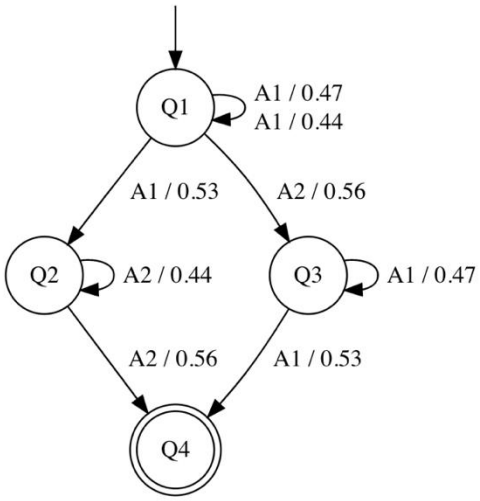

Рисунок 4 – Пример нечеткого автомата

Достоинством нечеткого автомата является его наглядность. А недостатком – сложность построения, при большом количестве состояний и переходов.

3. Композиция нечетких автоматов

Нечеткий автомат становится слишком большим и сложным, при большом количестве состояний и переходов между ними. Композиция нечетких автоматов лишена этих недостатков.

Композиция нечетких автоматов представляет собой множество нечетких автоматов.

Каждый логический оператор И или ИЛИ имеет свой нечеткий автомат. Также каждое начальное событие или риск-событие, это такое событие и риск-событие, которое не являются выходом ни одного из логических операторов И или ИЛИ, имеет свой нечеткий автомат.

В качестве алфавита для нечетких автоматов выступают события и риск-события. Причем, символы данного алфавита не только подаются на вход нечеткого автомата, но и когда автомат переходит в финальное состояние он генерирует новые символы для алфавита.

В качестве состояний для нечетких автоматов выступают ситуации и риск-ситуации.

Для построения множества нечетких матриц переходов используются вероятности событий и риск-событий.

Пример композиции нечетких автоматов представлен на рисунке 6. А структура событий и риск-событий, на основе которой построена композиция нечетких автоматов, представлена на рисунке 5.

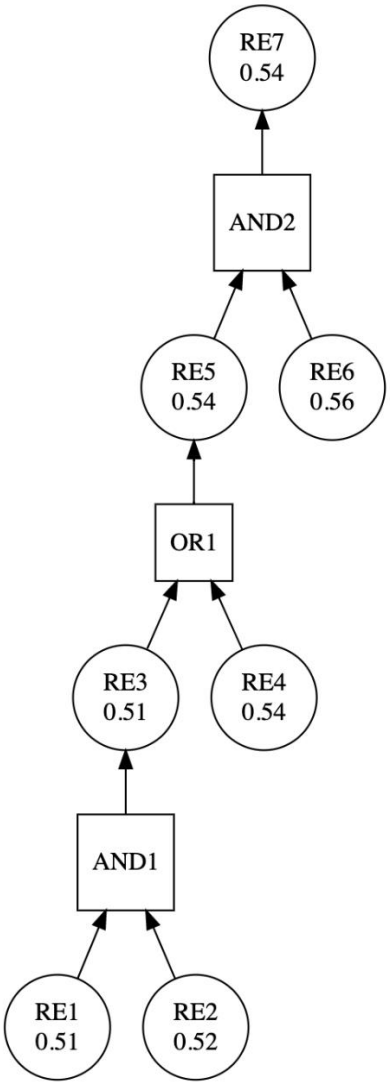

Рисунок 5 – Структура событий и риск-событий

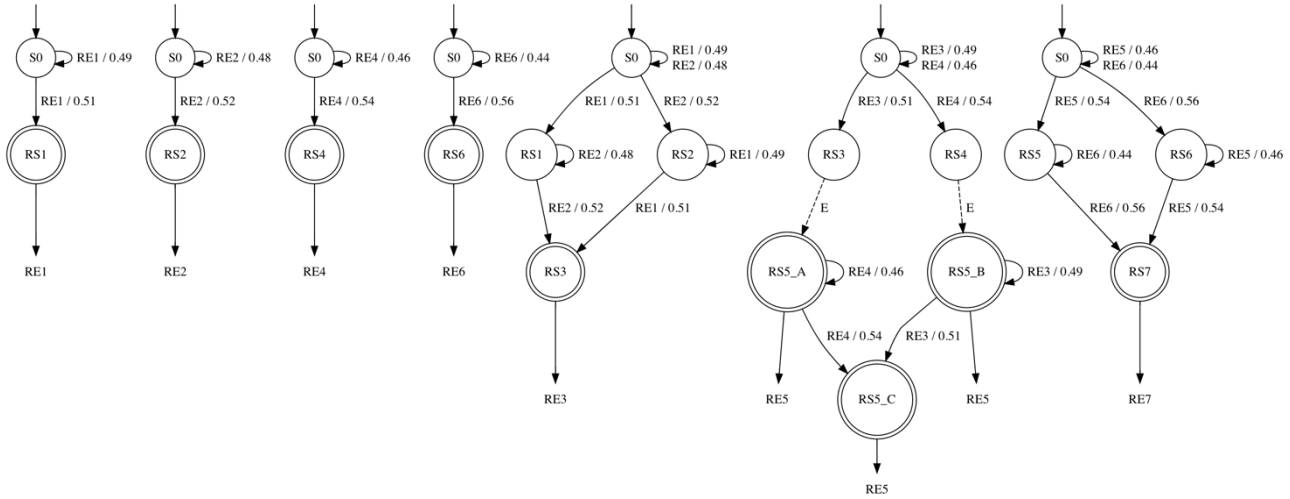

Рисунок 6 – Композиция нечетких автоматов для оценки ситуаций и риск-ситуаций

Композиция нечетких автоматов нужна для оценки ситуаций и риск-ситуаций, то есть, по сути, для оценки событий и риск-событий. А уже после оценки событий и риск-событий можно переходить к выбору сценариев управления рисками.

4. Способ построения композиции нечетких автоматов

Способ построения композиции нечетких автоматов состоит из следующих этапов:

Этап 1. Среди всех событий и риск-событий находятся начальные, это такие события и риск-события, которые не являются выходами логических операторов И или ИЛИ.

Этап 2. Для каждого начального события задается ситуация следующего вида:

$$S_{E_i} = \{E_i\} \tag{4.1}$$

Затем создается нечеткий автомат, в частности по формуле (2.2) задается множество состояний:

$$Q = \{S_0, S_{E_i}\}, \tag{4.2}$$

где S_0 – начальное состояние.

По формуле (2.3) задается вектор начальных состояний:

$$s = (1,0) \tag{4.3}$$

По формуле (2.4) задается вектор конечных состояний:

$$f = (0,1) \tag{4.4}$$

По формуле (2.5) задается множество входных символов (алфавит):

$$A = \{E_i\} \tag{4.5}$$

По формулам (2.6) и (2.7) задается множество нечетких матриц переходов:

$$T = \{T_{E_i}\} \tag{4.6}$$

$$T_{E_i} = \begin{pmatrix} 1 - P_{E_i} & P_{E_i} \\ 0 & 0 \end{pmatrix}$$

На рисунке 7 представлен пример нечеткого автомата для начального события.

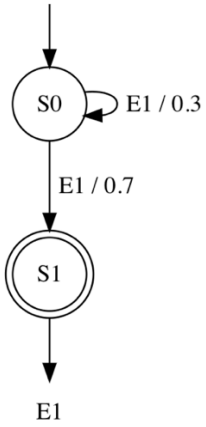

Рисунок 7 – Пример нечеткого автомата для начального события

Этап 3. Для каждого начального риск-события задается риск-ситуация следующего вида:

$$RS_{RE_i} = \{RE_i\} \tag{4.7}$$

Затем создается нечеткий автомат, в частности по формуле (2.2) задается множество состояний:

$$Q = \{S_0, RS_{RE_i}\}, \tag{4.8}$$

где S_0 – начальное состояние.

По формуле (2.3) задается вектор начальных состояний:

$$s = (1,0) \tag{4.9}$$

По формуле (2.4) задается вектор конечных состояний:

$$f = (0,1) \tag{4.10}$$

По формуле (2.5) задается множество входных символов (алфавит):

$$A = \{RE_i\} \tag{4.11}$$

По формулам (2.6) и (2.7) задается множество нечетких матриц переходов:

$$T = \{T_{RE_i}\} \tag{4.12}$$

$$T_{RE_i} = \begin{pmatrix} 1 - P_{RE_i} & P_{RE_i} \\ 0 & 0 \end{pmatrix}$$

На рисунке 8 представлен пример нечеткого автомата для начального риск-события.

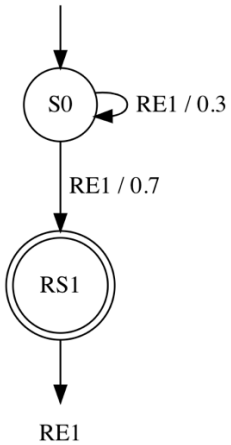

Рисунок 8 – Пример нечеткого автомата для начального риск-события

Этап 4. Для каждого логического оператора И задаются ситуации или риск-ситуации его входов и выхода.

Затем создается нечеткий автомат, в частности по формуле (2.2) задается множество состояний:

$$Q = \left\{S_0, \left(S_{E_i}|RS_{RE_i}\right), \left(S_{E_j}\middle|RS_{RE_j}\right), \left(S_{E_k}|RS_{RE_k}\right)\right\}, \quad (4.13)$$

где S_0 – начальное состояние; $S_{E_i}|RS_{RE_i}$ – ситуация или риск-ситуация для первого входа логического оператора И; $S_{E_j}|RS_{RE_j}$ – ситуация или риск-ситуация для второго входа логического оператора И; $S_{E_k}|RS_{RE_k}$ – ситуация или риск-ситуация для выхода логического оператора И.

По формуле (2.3) задается вектор начальных состояний:

$$s = (1,0,0,0) \quad (4.14)$$

По формуле (2.4) задается вектор конечных состояний:

$$f = (0,0,0,1) \quad (4.15)$$

По формуле (2.5) задается множество входных символов (алфавит):

$$A = \left\{(E_i|RE_i), \left(E_j\middle|RE_j\right)\right\} \quad (4.16)$$

По формулам (2.6) и (2.7) задается множество нечетких матриц переходов:

$$T = \left\{T_{E_i|RE_i}, T_{E_j|RE_j}\right\} \quad (4.17)$$

$$T_{E_i|RE_i} = \begin{pmatrix} 1 - P_{E_i|RE_i} & P_{E_i|RE_i} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 - P_{E_i|RE_i} & 0 & P_{E_i|RE_i} \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$T_{E_j|RE_j} = \begin{pmatrix} 1 - P_{E_j|RE_j} & 0 & P_{E_j|RE_j} & 0 \\ 0 & 1 - P_{E_j|RE_j} & 0 & P_{E_j|RE_j} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

На рисунке 9 представлен пример нечеткого автомата для логического оператора И.

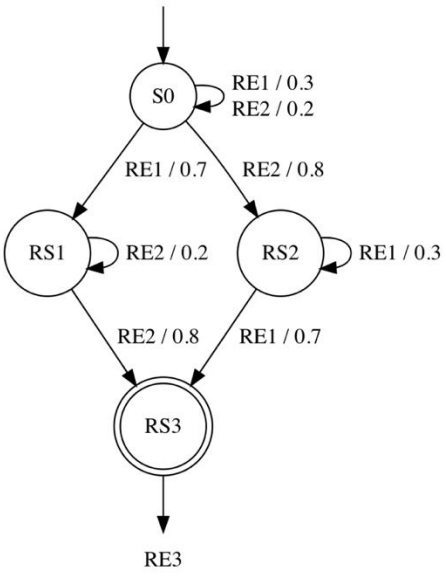

Рисунок 9 – Пример нечеткого автомата для логического оператора И

Этап 5. Для каждого логического оператора ИЛИ задаются ситуации или риск-ситуации его входов и выхода.

Затем создается нечеткий автомат, в частности по формуле (2.2) задается множество состояний:

$$Q = \begin{Bmatrix} S_0, \\ \left(S_{E_i} \middle| RS_{RE_i}\right), \\ \left(S_{E_j} \middle| RS_{RE_j}\right), \\ \left(S_{E_kA} \middle| RS_{RE_kA}\right), \\ \left(S_{E_kB} \middle| RS_{RE_kB}\right), \\ \left(S_{E_kC} \middle| RS_{RE_kC}\right) \end{Bmatrix}, \tag{4.18}$$

где S_0 – начальное состояние; $S_{E_i}|RS_{RE_i}$ – ситуация или риск-ситуация для первого входа логического оператора ИЛИ; $S_{E_j}|RS_{RE_j}$ – ситуация или риск-ситуация для второго входа логического оператора ИЛИ; $S_{E_kA}|RS_{RE_kA}$ – ситуация или риск-ситуация для выхода логического оператора ИЛИ, при срабатывание первого входа; $S_{E_kB}|RS_{RE_kB}$ – ситуация или риск-ситуация для выхода логического оператора ИЛИ, при срабатывание второго входа; $S_{E_kC}|RS_{RE_kC}$ – ситуация или риск-ситуация для выхода логического оператора ИЛИ, при срабатывание первого и второго входов.

По формуле (2.3) задается вектор начальных состояний:

$$s = (1,0,0,0,0,0) \tag{4.19}$$

По формуле (2.4) задается вектор конечных состояний:

$$f = (0,0,0,0,0,1) \tag{4.20}$$

По формуле (2.5) задается множество входных символов (алфавит):

$$A = \left\{(E_i|RE_i), \left(E_j\middle|RE_j\right), \varepsilon\right\} \tag{4.21}$$

По формулам (2.6) и (2.7) задается множество нечетких матриц переходов:

$$T = \left\{T_{E_i|RE_i}, T_{E_j|RE_j}\right\} \tag{4.22}$$

$$T_{E_i|RE_i} = \begin{pmatrix} 1-P_{E_i|RE_i} & P_{E_i|RE_i} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1-P_{E_i|RE_i} & P_{E_i|RE_i} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$T_{E_j|RE_j} = \begin{pmatrix} 1-P_{E_j|RE_j} & 0 & P_{E_j|RE_j} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1-P_{E_j|RE_j} & 0 & P_{E_j|RE_j} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\varepsilon = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

На рисунке 10 представлен пример нечеткого автомата для логического оператора ИЛИ.

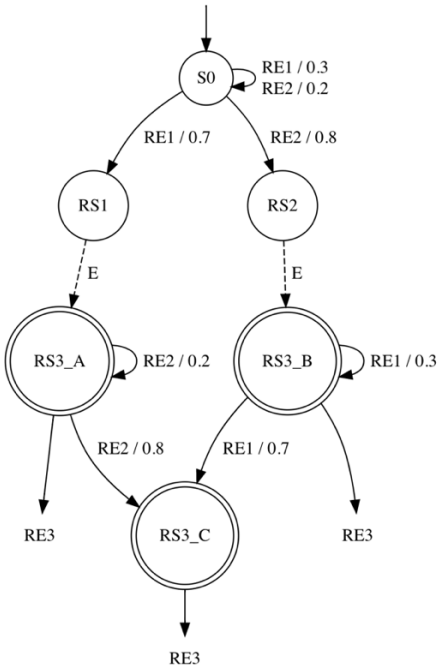

Рисунок 10 – Пример нечеткого автомата для логического оператора ИЛИ

5. Сценарий управления рисками

Сценарий управления рисками – это совокупность мероприятий. Сценарий можно описать с помощью следующей формулы:

$$SC = \{m | m \in M\}, \tag{5.1}$$

где SC – сценарий управления рисками; m – мероприятие; M – множество мероприятий.

Мероприятие – это совокупность действий нацеленных либо на событие или риск-событие, либо на логический оператор И или ИЛИ [1]. Мероприятие можно описать с помощью следующей формулы:

$$M_i = \langle T, D, C \rangle, \quad (5.2)$$

где M – мероприятие; i – номер мероприятия; T – тип мероприятия; D – описание мероприятия; C – стоимость мероприятия.

Мероприятие может иметь один из следующих типов:

- удаление события или риск-события;
- изменение события или риск-события;
- удаление логического оператора И или ИЛИ;
- изменение логического оператора И или ИЛИ.

Мероприятие, направленное на удаление события или риск-события, производит не только удаление события или риск-события, но и удаляет все логические операторы И или ИЛИ, которые используют в качестве входов или выхода данное событие или риск-событие. Данное удаление приводит к пересчету вероятностей наступления событий или риск-событий, которые раньше зависели от удаляемого события или риск-события.

Мероприятие, направленное на изменение события, может изменить вероятность наступления, данного события, а также сделать его риск-событием с добавление ущерба от наступления, данного риск-события. Данное изменение приводит к пересчету вероятностей наступления событий или риск-событий, которые раньше зависели от изменяемого события.

Мероприятие, направленное на изменение риск-событие, может изменить вероятность наступления, данного риск-события, а также изменить ущерб от наступления, данного риск-события. Также данное мероприятие может убрать полностью ущерб от наступления, данного риск-события, то есть сделать его обычным событием. Данное изменение приводит к пересчету вероятностей наступления событий или риск-событий, которые раньше зависели от изменяемого риск-события.

Мероприятие, направленное на удаление логического оператора И или ИЛИ, производит не только удаление логического оператора, но и приводит к пересчету вероятностей наступления событий или риск-событий, который раньше зависели от удаляемого логического оператора.

Мероприятие, направленное на изменение логического оператора И, может изменить входные и выходное события и риск-события. Также данное мероприятие может сделать его логическим оператором ИЛИ. Данное изменение приводит к пересчету вероятностей наступления событий или риск-событий, который раньше зависели от изменяемого логического оператора.

Мероприятие, направленное на изменение логического оператора ИЛИ, может изменить входные и выходное события и риск-события. Также данное мероприятие может сделать его логическим оператором И. Данное изменение приводит к пересчету вероятностей наступления событий или риск-событий, который раньше зависели от изменяемого логического оператора.

6. Способ выбора сценариев управления рисками

Способ выбора сценариев управления рисками состоит из следующих этапов:

Этап 1. Задается количество сценариев N, между которыми будет выбираться лучший сценарий управления рисками.

Этап 2. Среди всех событий и риск-событий находятся начальные, это такие события и риск-события, которые не являются выходами логических операторов И или ИЛИ.

Этап 3. Среди начальных событий и риск-событий находятся начальные активные события и риск-события.

Этап 4. Подаем все начальные активные события и риск-события на композицию нечетких автоматов, тем самым получая все активные события и риск-события.

Этап 5 Подсчитываем общий ущерб, как сумму ущербов всех активных риск-событий:

$$TC_{RE} = \sum C_{RE}, \tag{6.1}$$

где TC_{RE} – общий ущерб; C_{RE} – ущерб одного активного риск-события.

Этап 6. Формируем сценарий управления рисками SC, как пул случайных мероприятий и применяем его.

Этап 7. Подсчитываем стоимость сценария управления рисками по следующей формуле:

$$C_{SC} = \sum C_M, \tag{6.2}$$

где C_{SC} – стоимость сценария управления рисками; C_M – стоимость одного мероприятия из сценария управления рисками.

Этап 8. Повторяем этапы с 2 по 5. Если новый общий ущерб TC_{RE} меньше старого и стоимость сценария управления рисками C_{SC} не превышает новый общий ущерб, то помещаем кортеж $\langle SC, TC_{RE}, C_{SC}\rangle$ в массив всех сценариев управления рисками.

Этап 9. Если в массиве всех сценариев управления рисками достаточно сценариев, то есть совпадает с количеством сценариев N, то переходим к этапу 10, а иначе к этапу 6.

Этап 10. Среди сценариев из массива всех сценариев управления рисками выбираем тот, у которого наименьший общий ущерб и наименьшая стоимость:

$$\begin{gathered} TC_{RE} \to min \\ C_{SC} \to min \end{gathered} \tag{6.3}$$

Заключение

Сценарии управления рисками невозможно составить без идентификации, анализа и оценивания рисков. Поэтому, для идентификации, анализа и оценивания рисков используется композиция нечетких автоматов. А способ выбора сценариев управления рисками базируется на композиции нечетких автоматов.

Список литературы

1. Сеньков А.В. Управление рисками: интеллектуальные модели, методы, средства. – Смоленск: Универсум, 2016. – 217 с.: ил;
2. Основы анализа и управления риском в природной и технологической сферах. В.А. Акимов, В.В. Лесных, Н.Н. Радаев. - М.: Деловой экспресс, 2004. – 352 с.;
3. Остапенко Г.А., Карпеев Д.О., Плотников Д.Г., Батищев Р.В., Гончаров И.В., Маслихов П.А., Мешкова Е.А., Морозова Н.М., Рязанов С.А., Субботина С.В., Транин В.А. Риски распределенных систем: методики и алгоритмы оценки и управления // Информация и безопасность. 2010. Т. 13. № 4. С. 485-530;
4. Бадалова А.Г., Пантелеев А.В. Управление рисками деятельности предприятия // Учебное пособие для студентов, обучающихся по направлению подготовки 080200

"Менеджмент" (профиль "Производственный менеджмент") / Москва, 2016. (2-е издание);
5. Математическая логика и теория алгоритмов [Текст]: учеб. пособие / О. Ю. Агарева, Ю. В. Селиванов. – М.: МАТИ, 2011. – 80 с.;
6. Борисов В.В., Круглов В.В., Федулов А.С. Нечеткие модели и сети. – 2-е изд., стереотип. – М.: Горячая линия – Телеком, 2012. – 284 с.: ил;
7. Сперанский Д.В. Эксперементы с нечеткими автоматами // Автоматика и телемеханика. - 2015. - №2 С. 107-124;
8. Девятков В.В., Алфимцев А. Н. Нечеткая конечно-автоматная модель интеллектуального мультимодального интерфейса // Проблемы управления. - 2011. - №2 С. 69-77.

References

1. Senkov A.V. Risk management: intellectual models, methods, tools. - Smolensk: Universum, 2016 .-- 217 p.: Silt;
2. Basics of risk analysis and management in the natural and technological fields. V.A. Akimov, V.V. Lesnykh, N.N. Radaev. - M .: Business Express, 2004. - 352 p .;
3. Ostapenko G.A., Karpeev D.O., Plotnikov D.G., Batishchev R.V., Goncharov I.V., Maslikhov P.A., Meshkova E.A., Morozova N.M., Ryazanov S.A., Subbotina S.V., Tranin V.A. Risks of distributed systems: methodologies and algorithms for assessment and management // Information and Security. 2010. T. 13. No. 4. S. 485-530;
4. Badalova A.G., Panteleev A.V. Enterprise Risk Management // Textbook for students in the field of preparation 080200 "Management" (profile "Production Management") / Moscow, 2016. (2nd edition);
5. Mathematical logic and theory of algorithms [Text]: textbook. allowance / O. Yu. Agareva, Yu. V. Selivanov. - M .: MATI, 2011 .-- 80 s .;
6. Borisov V.V., Kruglov V.V., Fedulov A.S. Fuzzy models and networks. - 2nd ed., Stereotype. - M .: Hot line - Telecom, 2012 .-- 284 p.: Silt;
7. Speransky D.V. Experiments with fuzzy automata // Automation and Telemechanics. - 2015. - No. 2 S. 107-124;
8. Devyatkov VV, Alfimtsev AN N. Fuzzy finite-automaton model of an intelligent multimodal interface // Management Problems. - 2011. - No. 2 S. 69-77.

Сеньков А.В. Технический, организационный и инфраструктурный долг в разработке программного обеспечения и автоматизированных систем // Международный журнал информационных технологий и энергоэффективности. – 2020. – Т. 5 № 2(16) с. 36–42

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004

ТЕХНИЧЕСКИЙ, ОРГАНИЗАЦИОННЫЙ И ИНФРАСТРУКТУРНЫЙ ДОЛГ В РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И АВТОМАТИЗИРОВАННЫХ СИСТЕМ

Сеньков А.В.
Филиал ФГБОУ ВО «Национальный исследовательский университет «МЭИ» в г. Смоленске, Россия, (214013, г. Смоленск, Энергетический проезд, 1), e-mail: a.v.senkov@mail.ru.

Архитекторы и разработчики программного обеспечения и автоматизированных систем часто употребляют термин «технический» долг как некоторые отложенные задачи, которые могут привести к негативным последствиям в будущем. В настоящей статье предпринята попытка разобрать сущность долга, а также выполнить классификацию видов долга. Также проведена аналогия между управлением долгом и управлением рисками.

Ключевые слова: архитектура, программное обеспечение, автоматизированные системы, технический долг, организационный долг, риски.

TECHNICAL, ORGANIZATIONAL, AND INFRASTRUCTURAL DEBT IN A DEVELOPMENT OF SOFTWARE AND AUTOMATED SYSTEMS

Senkov A.V.
Smolensk Branch of the National Research University "Moscow Power Engineering Institute", Smolensk, Russia (214013, Smolensk, Energetichesky proezd, 1), e-mail: a.v.senkov@mail.ru.

Architects and developers of software and automated systems often use the term "technical" debt as some deferred tasks that could lead to negative consequences in the future. This article attempts to analyze the nature of debt, as well as classify types of debt. An analogy is also drawn between debt management and risk management.

Keywords: architecture, software, automated systems, technical debt, organizational debt, risks.

В 1992 году У.Каннингэм (Ward Cunningham) в своем докладе The WeCash Portfolio Management System на конференции OOPSLA (Conference on Object-Oriented Programming Systems, Languages, and Applications) ввёл понятие технического долга [1]. Следует отметить что именно в рамках доклада не выделяется именно технический долг. Вводится метафора долга в целом, который, однако, интерпретируется как незрелость программного кода. Поставка незрелого программного кода, или программного кода «с долгом» само по себе не приводит к негативным последствиям. Опасность возникает только в том случае, если долг не погашен. Он имеет свойство накапливаться, что в конечном итоге может привести к значительным трудозатратам, которые команда будет тратить на поддержку программного обеспечения.

В результате ряда выступлений и публикации ряда работ, например [2, 3], такой долг стал называться техническим поскольку действительно, причина его возникновения лежит в технической сфере, в коде программного обеспечения.

Что кардинально изменилось с 1992 года, когда такой долг был определен. В конце 80-х – начале 90-х годов получили значительное развитие и распространение многие методологии разработки программного обеспечения. В первую очередь, это гибкие методологии. Основная ценность в соответствии с Agile Manifest [4] в таких методология начинает переноситься на работающий продукт. Кроме того, значительное распространение получают прототипы, называемые MVP (minimum viewable product), которые, зачастую, разрабатываются малыми командами, после чего выводятся в продуктивную эксплуатацию.

Безусловно, в большинстве программных продуктов, прошедших путь через MVP, будет зафиксирован технический долг, однако, одновременно с продуктом, чаще всего, развивается и команда. Состав команды меняется, она пополняется новыми ролями, возникает потребность в новых функциях, изменении используемых организационных инструментов, подходов, практик и т.д.

При этом откладывание таких изменений в долгий ящик может также значительно усложнить работу по разработке и дальнейшей поддержке программного обеспечения. Каким образом можно поименовать такого рода недоработки?

Во-первых, эти недоработки заметны и могут быть идентифицированы, особенно подготовленным специалистом – специалистом с большим опытом работы в аналогичных командах / на аналогичных проектах.

Во-вторых, действия по устранению указанных недостатков действительно могут быть отложены, и, вероятно, это не приведет к значительным последствиям в краткосрочной перспективе.

В-третьих, сохранение этих недостатков принесет возможные потери в долгосрочной перспективе.

По аналогии с финансовыми долгами, предложенной изначально в [1], указанные недоработки в соответствии с их перечисленными особенностями, также можно отнести к одному из видов долга, однако, поскольку этот «долг» не относится непосредственно к программному коду, называть его техническим нецелесообразно. Справедливости ради, следует отметить, что в некоторых источниках, например [5, 6], такого вида задолженность относится к разряду технических, хотя непосредственно к программному коду отношения не имеет.

Как правило, автоматизированные системы рассматривают в виде триплета [7], включающего программное обеспечение, аппаратное обеспечение и организационное обеспечение. Если рассматривать долг сквозь призму видов обеспечения автоматизированных систем (частей автоматизированных систем), то и долг может быть разделен исходя из принадлежности к одной из таких частей. Исходя из этого, целесообразно сформировать следующую классификацию долга.

1. Технический долг. Поскольку термин «технический долг» можно считать прочно вошедшим в обиход специалистов по программной инженерии, целесообразно этим термином обозначать любой долг, относящийся к программному обеспечению, хотя более точным определением, вероятно, стало бы определение «Программный долг».

2. Инфраструктурный или аппаратный долг. Ранее явно не выделявшийся вид долга, к которому можно в зависимости от конкретного проекта или команды относить или долг по улучшению аппаратного обеспечения, на котором развертывается или используется автоматизированная система. К такого рода долгу можно отнести недостаточную обеспеченность серверными мощностями / средствами хранения информации и т.д. Например, недостаточное резервирование серверных мощностей значительно повышает вероятность отказа системы с сопутствующими невосстановимыми потерями.
3. Организационный долг – совокупность допущений и ограничений, принятых в организации и/или команде, которые в результате развития организации/команды могут значительным образом повлиять на стабильность организации/команды или разрабатываемой ими системы. Организационный долг является наиболее сложным для идентификации не только по причине того, что для идентификации нужен специалист с широким кругозором в смысле организации труда, но и ввиду отсутствия точных рецептов организации труда в такой отрасли как разработка автоматизированных систем.

Помимо рассмотренной классификации по частям автоматизированной системы, долг может также быть классифицирован по ролям, накапливающим этот долг. Заранее определимся что будем рассматривать только роли, относящиеся непосредственно к разработке автоматизированных систем. Кроме того, возьмем расширенный перечень ролей, характерных для различных методологий разработки автоматизированных систем [8, 9, 10], характерных для большинства организаций и команд.

Рассмотрим возможность указанных ролей накапливать различные виды долга (таблица 1).

Таблица 1 – Роли и их возможности по накоплению различных видов долга

№ п/п	Роль	Технический долг	Инфра-структур-ный долг	Организа-ционный долг
1	**2**	**3**	**4**	**5**
1	Директор, директор по информационным технологиям (CIO) – руководитель	-	+	++
2	Технический директор (CTO)	++	++	+
3	Директор по инфраструктуре	-	++	++
4	Директор по информационной безопасности	+	+	++
5	Операционный директор (COO, Scrum Master)	-	-	++
6	Руководитель продукта (Product Owner)	+	+	+
7	Руководитель программы (Program Manager, Product Development Manager, System Engineering Manager и др.)	+	+	++

№ п/п	Роль	Технический долг	Инфра-структур-ный долг	Организа-ционный долг
1	**2**	**3**	**4**	**5**
8	Руководитель разработки (Team Lead)	++	+	+
9	Архитектор (Enterprise Architect, Solution Architect и др.)	++	++	-
10	Аналитик (бизнес, системный, не продуктовый)	+	-	-
11	Разработчик	++	-	-
12	Специалист по качеству (Тестировщик)	+	-	-
13	Системный администратор	-	+	-
14	UX	+	-	+
15	Технический писатель	+	-	+

В таблице:

«-» - означает что для роли не характерно накопление соответствующего вида долга;

«+» - означает что роль ограниченно может накапливать соответствующий долг

«++» - означает что роль в значительной степени может накапливать соответствующий долг.

Таким образом, различные роли могут по-разному влиять на накопление долга различного вида.

Кроме того, при рассмотрении причин накопления долга, можно отметить что организации/команды на различных этапах своего развития могут накапливать долг разного рода в разной степени. Этот вопрос, очевидно, касается траекторий развития организаций/команд и должен быть проработан отдельно.

Однако, каким образом можно пробовать управлять различного рода долгом. Если долг может нести для организации или команды потери в определенной перспективе, то можно провести аналогию между долгом и рисками. При этом, для рисков разработано достаточно большое количество методов, моделей и средств для управления ими [11, 12, 13, 14]. Если рассматривать управление долгом с точки зрения процесса управления рисками, представленного в [13], многие задачи управления долгом становятся очевидными (см. рисунок 1). Однако, каждый из этапов процесса управления долгом должен быть рассмотрен и детализирован отдельно.

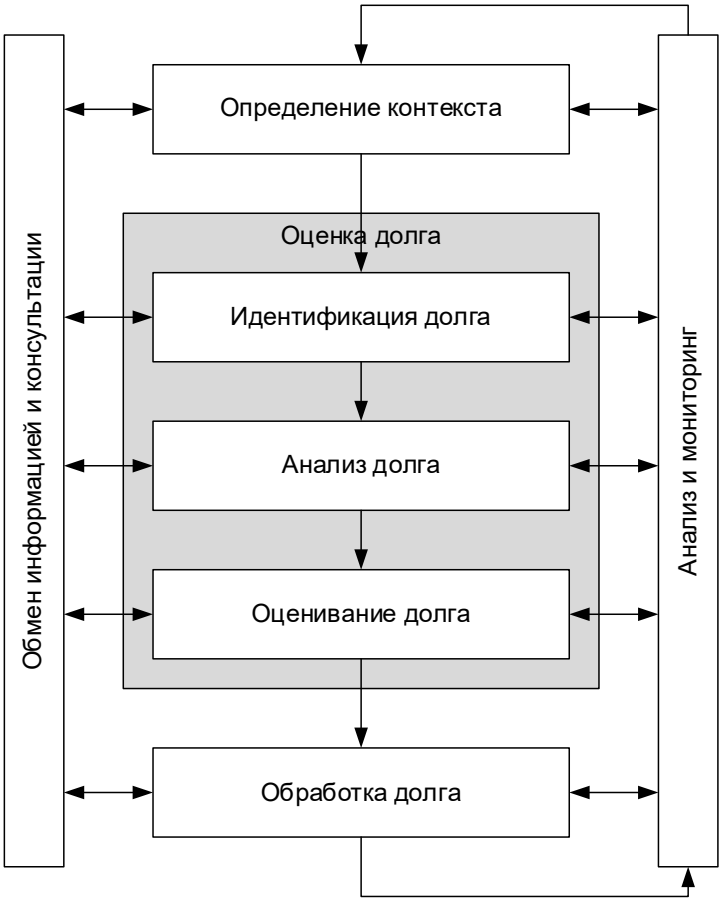

Рисунок 1 – Обобщенный процесс управления долгом

Таким образом, долг, накапливаемый в рамках процесса разработки автоматизированных систем можно разделить на технический (программный), инфраструктурный (аппаратный) и организационный. Различные роли, участвующие в разработке автоматизированных систем, в силу своих обязанностей могут допускать накопление различного рода долга в разной степени. Управление долгом можно рассматривать по аналогии с управлением рисками.

Список литературы

1. Cunningham, W.: The WyCash Portfolio Management System. In: Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA 1992, pp. 29–30. ACM, New York (1992);
2. Fowler M. TechnicalDebt [сайт]. URL: https://www.martinfowler.com/bliki/TechnicalDebt.html (дата обращения: 11.06.2020);
3. Yli-Huumo J., Maglyas A., Smolander K. (2014) The Sources and Approaches to Management of Technical Debt: A Case Study of Two Product Lines in a Middle-Size Finnish Software Company. In: Jedlitschka A., Kuvaja P., Kuhrmann M., Männistö T., Münch J., Raatikainen M. (eds) Product-Focused Software Process Improvement. PROFES 2014. Lecture Notes in Computer Science, vol 8892. Springer, Cham

4. Galbraith K. Two Kinds of Tech Debt and How to Pay It Down [сайт] URL: https://blog.kylegalbraith.com/2018/10/22/two-kinds-of-tech-debt-and-how-to-pay-it-down/ (дата обращения: 11.06.2020);
5. Agile-манифест разработки программного обеспечения [сайт] URL: https://agilemanifesto.org/iso/ru/manifesto.html (дата обращения: 11.06.2020);
6. Yli-Huumo J., Maglyas A., Smolander K. (2016) The Effects of Software Process Evolution to Technical Debt—Perceptions from Three Large Software Projects. In: Kuhrmann M., Münch J., Richardson I., Rausch A., Zhang H. (eds) Managing Software Process Evolution. Springer, Cham DOI: 10.1007/978-3-319-31545-4_15
7. Информационные технологии : учеб. пособие / Г.Н. Исаев. – М. : Издательство «Омега-Л», 2012.
8. Руководство к Своду знании по управлению проектами (Руководство PMBOK®) -- Шестое издание - PMI, 2017.
9. A. Stellman, J. Greene Learning Agile. – O'Reilly Media, Inc., 2014
10. J. Sutherland, J.O. Coplien A Scrum Book. – Pragmatic Bookshelf., 2019
11. Королев В.Ю., Беннинг В.Е., Шоргин С.Я. Математические основы теории риска: Учебн. пособ. – 2–е изд., перераб. и доп. – М.: Физматлит, 2011.
12. ISO 31000:2009 – Risk management Principles and guidelines
13. Борисов В.В., Сеньков А.В. Интеллектуальное управление рисками в сложных организационно-технических системах // Информационные технологии, 2011, № 10. – С. 47–51.
14. Сеньков А.В. Управление рисками: интеллектуальные модели, методы, средства. – Смоленск: Универсум, 2016

References

1. Cunningham, W.: WyCash Portfolio Management System. In the book: Supplement to the materials "Object-oriented programming systems, languages and applications", OOPSLA 1992, p. 29-30. AFM, New York (1992);
2. Fowler M. TechnicalDebt [site]. URL: https://www.martinfowler.com/bliki/TechnicalDebt.html (accessed date: 06/11/2020);
3. Yli-Huumo J., Maglyas A., Smolander K. (2014) Sources and approaches to technical debt management: a case study of two product lines in a medium-sized Finnish software company. In: Jedlitschka A., Kuvaja P., Kuhrmann M., Männistö T., Münch J., Raatikainen M. (eds) Improving the product-oriented software development process. PROFES 2014. Lecture Notes in Computer Science, Volume 8892. Springer, Cham
4. Galbraith K. Two types of technical debt and how to pay it off [site] URL: https://blog.kylegalbraith.com/2018/10/22/two-kinds-of-tech-debt-and-how-pay- down / (circulation date: 06/11/2020);
5. Agile-manifest of software development [site] URL: https://agilemanifesto.org/iso/ru/manifesto.html (accessed: 06/11/2020);
6. Juli-Uumo J., Maglyas A., Smolander K. (2016) The influence of the evolution of software processes on technical debt - the perception of three major software projects. In the book: Kurmann M., Münch J., Richardson I., Rausch A., Zhang H. (eds.) Software Development Management. Springer, Cham DOI: 10.1007 / 978-3-319-31545-4_15

7. Information technology: textbook. allowance / G.N. Isaev. - M.: Publishing house "Omega-L", 2012.
8. Guide to the Project Management Body of Knowledge (PMBOK® Guide) - Sixth Edition - PMI, 2017.
9. A. Stellman, J. Greene Learning Agile. – O'Reilly Media, Inc., 2014
10. J. Sutherland, J.O. Coplien A Scrum Book. – Pragmatic Bookshelf., 2019
11. Korolev V.Yu., Benning V.E., Shorgin S.Ya. Mathematical foundations of risk theory: Textbook. benefits - 2nd ed., Rev. and add. - M.: Fizmatlit, 2011.
12. ISO 31000: 2009 - Principles and guidelines for risk management
13. Borisov V.V., Senkov A.V. Intelligent risk management in complex organizational and technical systems // Information Technologies, 2011, No. 10. - P. 47-51.
14. Senkov A.V. Risk management: intellectual models, methods, tools. - Smolensk: Universum, 2016.

Борисов В.В., Арбузов А.Д., Колягина С.Д. Идентификация и анализ устойчивости кластеров социотехнических систем на основе нечеткого когнитивного подхода // Международный журнал информационных технологий и энергоэффективности. – 2020. – Т. 5 № 2(16) с. 43–52

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004.82

ИДЕНТИФИКАЦИЯ И АНАЛИЗ УСТОЙЧИВОСТИ КЛАСТЕРОВ СОЦИОТЕХНИЧЕСКИХ СИСТЕМ НА ОСНОВЕ НЕЧЕТКОГО КОГНИТИВНОГО ПОДХОДА

[1]Борисов В.В., [2]Арбузов А.Д., [3]Колягина С.Д.
[1,2]Филиал ФГБОУ ВО «Национальный исследовательский университет «МЭИ» в г. Смоленске, Россия, (214013, г. Смоленск, Энергетический проезд, 1), e-mail: [1]vbor67@mail.ru, [2]alex_a97@mail.ru
[3]Военная академия ВПВО ВС РФ

Предложена постановка и решение задачи идентификации и анализа устойчивости кластеров социотехнических систем на основе нечеткого когнитивного подхода. Для идентификации кластеров обоснована совокупность показателей, основанная на нечетких отношениях взаимовлияния между объектами в этих системах. Описана процедура кластеризации объектов и идентификации кластеров социотехнической системы, позволяющая в итоге оценить нечеткие степени принадлежности этих объектов к кластерам, а также определить их характеристические особенности по результатам анализа свойств объектов, относящихся к каждому выделенному кластеру. Для анализа устойчивости кластеров объектов в социотехнических системах предложен оригинальный способ, заключающийся в оценке результатов согласования нечетких отношений взаимовлияния между объектами каждого из идентифицированных кластеров социотехнической системы. Представлены результаты использования предлагаемого подхода для идентификации и анализа устойчивости энергетической системы Смоленского региона.

Ключевые Социотехническая система, идентификация кластеров и анализ устойчивости, нечеткая когнитивная модель, нечеткое отношение взаимовлияния.

IDENTIFICATION AND ANALYSIS OF SUSTAINABILITY OF CLUSTERS OF SOCIOTECHNICAL SYSTEMS BASED ON A FUZZY COGNITIVE APPROACH

[1]Borisov V.V., [2]Arbuzov A.D., [3]Kolyagina S.D.
Smolensk Branch of the National Research University "Moscow Power Engineering Institute", Smolensk, Russia (214013, Smolensk, Energetichesky proezd, 1), e-mail: [1]vbor67@mail.ru, [2]alex_a97@mail.ru
[3]Military academy of air defense of the Armed Forces of the Russian Federation

The formalization and solution of the problem of identifying and analyzing the sustainability of clusters of socio-technical systems based on a fuzzy cognitive approach is proposed. To identify clusters, a set of indicators based on fuzzy relationships of mutual influence between objects in these systems is justified. The procedure of clustering objects and identifying clusters of a sociotechnical system is described. It allows to evaluate the fuzzy degrees of belonging of these objects to clusters, as well as to determine their characteristic features by analyzing the properties of objects related to each selected cluster. To analyze the sustainability of clusters of objects in sociotechnical systems, an original method is proposed. It consists in assessing the results of coordination of fuzzy mutual influence relationships between objects of each of the identified clusters of a sociotechnical system. The results of using the proposed approach to identify and analyze the stability of the energy system of the Smolensk region are presented.

Keywords: Sociotechnical system, cluster identification and sustainability analysis, fuzzy cognitive model, fuzzy relationship of mutual influence.

Особенностями социотехнических систем является то, что их характеризуют: с одной стороны объединяющие свойства (например: общая сырьевая база; локализация на отдельной территории; технологическое, коммерческое и логистическое взаимодействие объектов; профильная специализация; синергетический эффект и повышение конкурентоспособности в результате взаимодействия); а с другой стороны, самостоятельность и активность входящих в эти системы разнородных объектов, а также разнотипность отношений между ними в зависимости от конкретных целей и задач их функционирования и развития [1-3].

Это приводит не только к проблеме возможности использования типовых подходов для решения задачи идентификации и анализа устойчивости кластеров объектов, входящих в эти социотехнические системы [4-6], но и вообще, к сложности постановки самой этой задачи для систем указанного класса.

Подходом, ориентированным на решение указанной проблемы, является нечеткий когнитивный подход, позволяющий трактовать все взаимосвязи и взаимозависимости между объектами социотехнической системы в виде не-четких отношений взаимовлияния (то есть, каузальных, причинно-следственных отношений), и, таким образом, применить для анализа устойчивости социотехнических систем и их кластеров аппарат нечеткой каузальной алгебры, описанный в [7].

В рамках решения задач идентификации и анализа устойчивости кластеров объектов в социотехнических системах в данной статье:

- во-первых, для идентификации кластеров объектов в социотехнических системах обосновывается совокупность показателей, основанная на нечетких отношениях взаимовлияния между этими объектами, и сформированная исходя из анализа построенной нечеткой когнитивной модели;
- во-вторых, для анализа устойчивости кластеров объектов в социотехнических системах предлагается использовать результаты оценки согласованности нечетких отношений взаимовлияния между этими объектами.

1. Постановка задачи

Пусть имеется множество объектов $A=\{a_i \mid i=1..I\}$ социотехнической системы, связанных друг с другом различными производственно-технологическими, коммерческими, территориальными, логистическими и иными взаимодействиями, трактуемыми с точки зрения нечеткого отношения взаимовлияния:

$$R=\left\{\left(r(a_i,a_j)/(a_i,a_j)\right) \mid r(a_i,a_j)\in[-1,1],\, a_i,a_j\in A\right\},$$

такого, что $R:(a_i,a_j)\to[-1,1],\forall a_i,a_j\in A$, т.е. ставит в соответствие каждой паре объектов $(a_i,a_j)\in A\times A$ значение (степень) взаимовлияния $r(a_i,a_j)\in[-1,1]$.

Требуется:

во-первых, обосновать совокупность показателей для идентификации кластеров социотехнической системы, основанную на нечетком отношении взаимовлияния между объектами этой системы:

$$P = \{ p_k \mid k = 1..K \};$$

во-вторых, на основе обоснованной совокупности показателей выполнить кластеризацию объектов и осуществить идентификацию кластеров социотехнической системы

$$C = \{ c_l \mid l = 1..L \};$$

в-третьих, выполнить анализ устойчивости кластеров социотехнической системы на основе результатов оценки согласованности нечеткого отношения взаимовлияния между объектами кластеров и системы в целом.

2. Обоснование совокупности показателей для идентификации кластеров социотехнической системы на основе нечеткого отношения взаимовлияния между ее объектами

Ранее проведенные исследования [8] позволили определить целесообразность применения нечеткого когнитивного подхода для обоснования совокупности показателей для идентификации кластеров социотехнической системы.

Процедура обоснования этой системы показателей заключается в следующем.

Шаг 1. Формирование нечеткой когнитивной модели (НКМ), концепты которой соответствует объектам социотехнической системы $A = \{ a_i \mid i = 1..I \}$.

Шаг 2. Задание (либо экспертами, либо в результате экспериментов) нечеткого отношения взаимовлияния между концептами НКМ:

$$R = \left\{ \left(r(a_i, a_j) / (a_i, a_j) \right) \mid r(a_i, a_j) \in [-1, 1],\ a_i, a_j \in A \right\}.$$

Отношение R представим в виде матрицы смежности $\mathbf{R} = \left\| r_{ij} \right\|_{I \times I}$, где r_{ij} – степень влияния концепта-источника a_i на концепт-приемник a_j, I – число концептов модели.

Шаг 3. Преобразование матрицы $\mathbf{R}$, значения которой представлены в диапазоне $[-1, 1]$, в матрицу неотрицательных значений $\mathbf{Q} = \left\| q_{ij} \right\|_{2I \times 2I}$ по правилам:

$$\text{Если } r_{ij} > 0, \text{ То } q_{2i-1,\,2j-1} = r_{ij} \text{ И } q_{2i,\,2j} = r_{ij},$$
$$\text{Если } r_{ij} < 0, \text{ То } q_{2i-1,\,2j-1} = -r_{ij} \text{ И } q_{2i,\,2j} = -r_{ij}.$$

Шаг 4. Согласование матрицы $\mathbf{Q}$ на основе ее транзитивного замыкания:

$$\widehat{\mathbf{Q}} = \mathbf{Q} \vee \mathbf{Q}^2 \vee \mathbf{Q}^3 \vee \ldots,$$

где матрицы $\mathbf{Q}, \mathbf{Q}^2, \mathbf{Q}^3, \ldots$ формируются на основе max-prod-композиции, а в качестве «$\vee$» используется операция max.

Если не удается достичь транзитивного замыкания матрицы $\mathbf{Q}$, то модель считается неустойчивой, анализируются причины ее неустойчивости, и, если необходимо, определяются возможности приведения модели в устойчивое состояние.

Шаг 5. Преобразование транзитивно замкнутой матрицы $\widehat{\mathbf{Q}}$ в матрицу $\mathbf{R}' = \left\| (r'_{ij},\ \overline{r}'_{ij}) \right\|_{I \times I}$, описывающую согласованное нечеткое отношение взаимовлияний между объектами социотехнической системы и состоящую из положительно-отрицательных пар элементов, образованных по правилам:

$$r'_{ij} = \max\left(q_{2i-1,\,2j-1},\, q_{2i,\,2j}\right),\ \overline{r}'_{ij} = -\max\left(q_{2i-1,\,2j},\, q_{2i,\,2j-1}\right).$$

Шаг 6. Выбор совокупности показателей для идентификации кластеров социотехнической системы из множества системных показателей, основанных на согласованном нечетком отношении взаимовлияния между объектами этой системы.

Из всего множества системных показателей, основанных на согласованном нечетком отношении взаимовлияния [7], для идентификации кластеров социотехнической системы по результатам проведенных исследований обоснован выбор совокупности следующих показателей:

- *показатель воздействия объекта на социотехническую систему*

$$p_1(a_i) = \frac{1}{I}\sum_{j=1}^{I}\left(\operatorname{sign}\left(r'_{ij} + \overline{r}'_{ij}\right)\max\left(\left|r'_{ij}\right|, \left|\overline{r}'_{ij}\right|\right)\right),\ i = 1..\,I;$$

- *показатель воздействия социотехнической системы на объект*

$$p_2(a_i) = \frac{1}{I}\sum_{j=1}^{I}\left(\operatorname{sign}\left(r'_{ji} + \overline{r}'_{ji}\right)\max\left(\left|r'_{ji}\right|, \left|\overline{r}'_{ji}\right|\right)\right),\ i = 1..\,I;$$

- *показатель согласованного влияния объекта на социотехническую систему*

$$p_3(a_i) = \frac{1}{I}\sum_{j=1}^{I}\frac{\left|r'_{ij} + \overline{r}'_{ij}\right|}{\left|r'_{ij}\right| + \left|\overline{r}'_{ij}\right|},\ i = 1..\,I.$$

3. Кластеризация объектов и идентификация кластеров социотехнической системы

Для кластеризации объектов социотехнической системы, в соответствии с обоснованной в предыдущем разделе совокупностью показателей, предлагается использовать *иерархический агломеративный метод* (метод полной связи) [9], который позволяет построить дендрограмму разбиения объектов, иллюстрирующую зависимость качества кластеризации от числа кластеров.

Для определения же наиболее рационального числа кластеров в дополнение к иерархическому агломеративному методу предлагается использовать *метод силуэтов* [10]. При этом среднее значение ширины всех силуэтов рассматривается как показатель качества кластеризации.

После фиксации количества кластеров, соответствующих наиболее рациональному разбиению объектов на кластеры (т.е. при максимальном значении показателя качества кластеризации), предлагается воспользоваться методом кластерного анализа *C-means*, который позволяет определить нечеткие степени принадлежности этих объектов к кластерам [11].

Идентификация же кластеров социотехнической системы заключается в анализе свойств (производственно-технологических, коммерческих, территориальных, логистических и иных) объектов, относящихся к каждому выделенному кластеру, и в определении их характеристических особенностей.

4. Анализ устойчивости кластеров социотехнической системы

Рассмотренный в данной статье нечеткий когнитивный подход к идентификации кластеров социотехнических систем, во многом, определяет и способ, который целесообразно использовать для анализа устойчивости идентифицированных кластеров социотехнической системы.

Предлагаемый способ заключается анализе результатов согласования (транзитивного замыкания) нечетких отношений взаимовлияния между объектами каждого из идентифицированных кластеров социотехнической системы.

При этом вывод о степени устойчивости какого-либо кластера социотехнической системы делается на основании следующего правила: *степень устойчивости кластера социотехнической системы зависит от отношения числа итераций процедуры транзитивного замыкания к числу объектов этого кластера. Чем оно меньше, тем более устойчивыми являются этот кластер.*

Данное правило справедливо и для анализа устойчивости социотехнической системы в целом (см. раздел 2).

5. Пример идентификации и анализа устойчивости кластеров социотехнических систем

Ниже представлены результаты использования предлагаемого подхода для идентификации и анализа устойчивости энергетической системы Смоленского региона.

В результате предварительного исследования определено множество объектов этой социотехнической системы, построена НКМ, и на основе согласованного нечеткого отношения взаимовлияния между ее объектами рассчитаны значения показателей для кластеризации выявления кластеров энергетической системы Смоленского региона (см. таблицу 1).

На рисунке 1 представлена дендрограмма, полученная в результате применения иерархического агломеративного метода для кластеризации объектов энергетической системы Смоленского региона, а на рисунке 2 – график зависимости выбранного показателя качества кластеризации от количества кластеров.

Показатель качества кластеризации принимает наибольшие значения при разбиении объектов на 6 (0.446) и на 9 кластеров (0.450).

Зафиксируем разбиение объектов на 6 кластеров:

c_1: $\{a_1, a_{13}, a_{19}, a_{20}\}$; c_2: $\{a_7, a_{21}\}$; c_3: $\{a_2, a_3\}$;
c_4: $\{a_4, a_5, a_6, a_{10}, a_{12}, a_{17}\}$; c_5: $\{a_8, a_{11}, a_{14}, a_{15}, a_{18}, a_{22}\}$; c_6: $\{a_9, a_{16}\}$.

Для зафиксированного числа кластеров применим метод C-means со значением коэффициента размытости (weighting exponent) 2, и значением параметра остановки 0.001.

На рисунке 3 показано расположение объектов и центров выявленных кластеров в пространстве обоснованных показателей P. В таблице 2 представлена матрица степеней принадлежности объектов к соответствующим кластерам.

Полученные результаты позволяют специалистам предметной области выполнить идентификацию кластеров энергетической системы Смоленского региона, заключающуюся в углубленном анализе свойств относящихся к ним объектов.

Таблица 1 – Значения показателей для идентификации энергетических кластеров Смоленского региона

Объекты	Десногорский энергетический колледж	Филиал «НИУ «МЭИ» в г. Смоленске	Смоленская АЭС	Смоленская ГРЭС	Смоленская ТЭЦ-2	Дорогобужская ТЭЦ	Смоленсктеплосеть	Дорогобужское тепловое хозяйство	ООО «Гидрострой	ЗАО «Фирма Энерго+»	ГК «Турбопар»	ООО «ЭнергоМонтаж Автоматика-ЭП»	ФГУП «СПО «Аналитприбор»	«АБО Арматура»	АО «Стройкомплект-Эмаль»	ООО «Глубур-Сервис»	ООО «ТД «Автоматика»	ОАО «Дорогобужкотломаш»	ОАО «Смоленскэнерго»	Филиал «СмоленскАтом ЭнергоСбыт»	ООО «Русэлпром- СЭЗ»	ООО «ЭнергоПромМаркет»
	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}	a_{20}	a_{21}	a_{22}
$p_1(a_i)$	0.329	0.448	0.505	0.218	0.209	0.215	0.230	0.145	0.123	0.137	0.143	0.142	0.334	0.145	0.122	0.023	0.162	0.162	0.351	0.284	0.202	0.172
$p_2(a_i)$	0.309	0.330	0.307	0.237	0.272	0.223	0263	0.183	0.114	0.204	0.199	0.175	0.237	0.153	0.096	0.106	0.165	0.099	0.309	0.330	0.271	0.223
$p_3(a_i)$	0.678	0.692	0.659	0.512	0.491	0.480	0.675	0.612	0.348	0.474	0.649	0.420	0.645	0.667	0.623	0.376	0.422	0.643	0.649	0.552	0.715	0.610

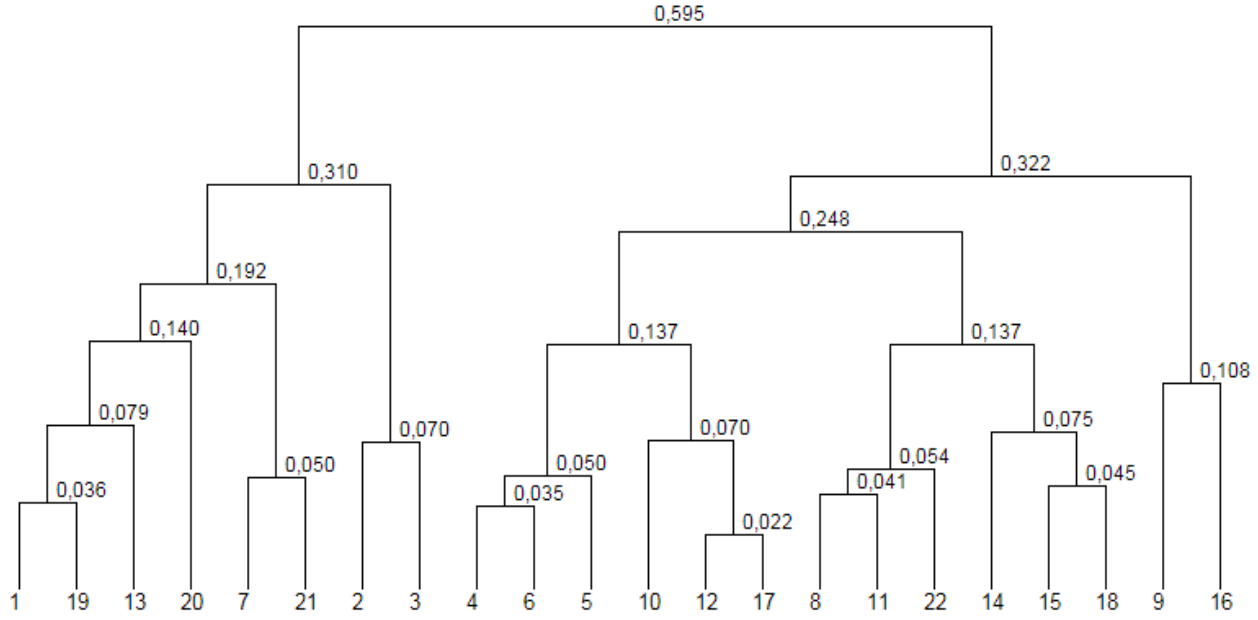

Рисунок 1 – Дендрограмма кластеризации объектов энергетической системы Смоленской области (иерархический алгоритм)

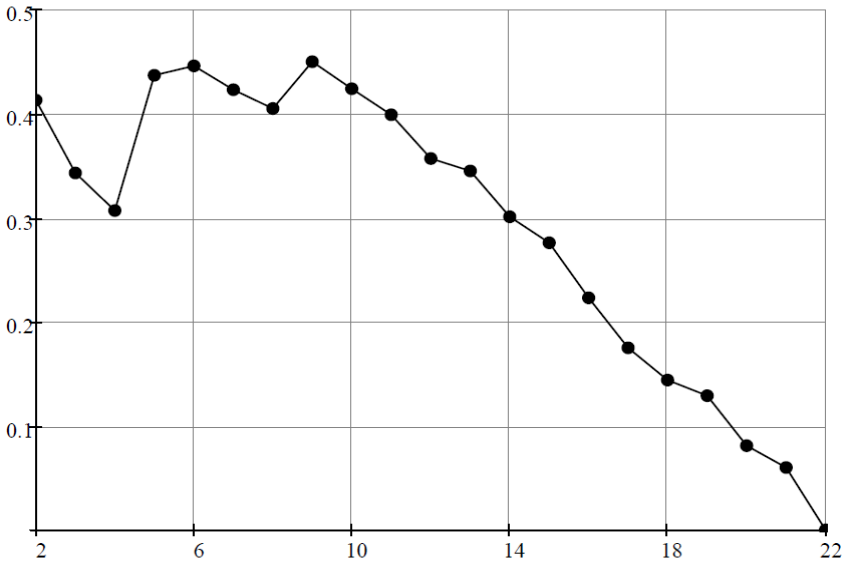

Рисунок 2 – Зависимость качества кластеризации от количества кластеров

Как отмечено в разделе 4, в основе вывода об устойчивости энергетической системы Смоленского региона лежит анализ результатов согласования нечетких отношений взаимовлияния между объектами каждого из идентифицированных кластеров социотехнической системы, а также системы в целом.

Так, при выполнении процедуры согласования нечеткого отношения взаимовлияния построенной НКМ транзитивное замыкание нечеткой матрицы положительных связей **Q** сошлось за 4 итерации ($<< I$), что свидетельствует о существенной устойчивости энергетической системы Смоленского региона. Такие же выводы можно сделать относительно всех выделенных кластеров системы.

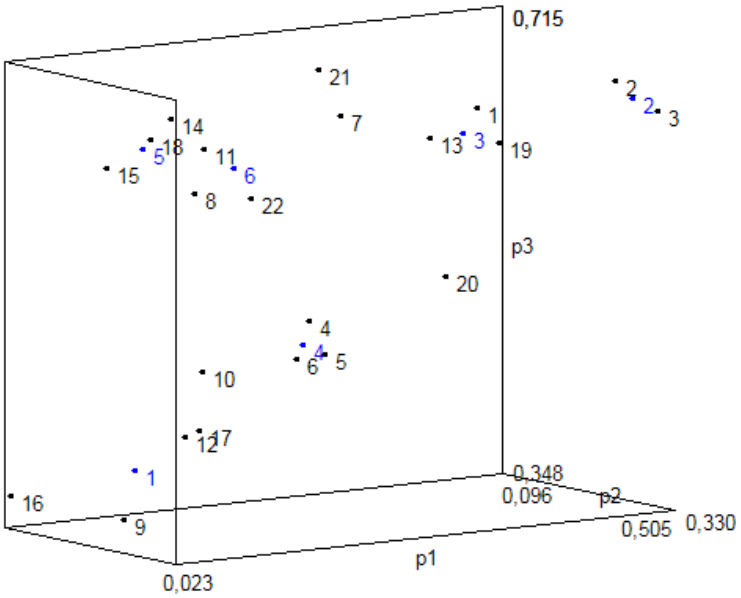

Рисунок 3 – Результаты кластеризации объектов энергетической системы Смоленского региона

Таблица 2 – Матрица степеней принадлежностей объектов к соответствующим кластерам энергетической системы Смоленского региона

Кластеры	Объекты																					
	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}	a_{20}	a_{21}	a_{22}
1	.005	.004	.005	.011	.024	.022	.031	.022	**.875**	.277	.010	**.752**	.019	.019	.016	**.680**	**.681**	.007	.005	.055	.038	.011
2	.035	**.908**	**.930**	.004	.008	.005	.055	.009	.008	.019	.005	.010	.084	.011	.005	.021	.013	.003	.050	.097	.063	.006
3	**.917**	.060	.038	.010	.020	.011	.317	.024	.014	.041	.016	.019	**.721**	.028	.011	.035	.025	.008	**.902**	**.348**	.248	.021
4	.014	.010	.010	**.947**	**.899**	**.928**	.102	.052	.051	**.452**	.023	.138	.059	.035	.020	.106	.181	.011	.016	.294	.098	.043
5	.010	.007	.007	.010	.017	.013	.110	.202	.027	.090	.091	.039	.044	**.604**	**.888**	.085	.049	**.932**	.010	.067	.142	.048
6	.019	.010	.010	.019	.032	.021	**.386**	**.692**	.025	.121	**.855**	.042	.075	.303	.060	.073	.051	.039	.017	.139	**.412**	**.870**

Заключение

В статье предложена оригинальная постановка и решение задачи идентификации и анализа устойчивости кластеров социотехнических систем на основе нечеткого когнитивного подхода, позволяющего трактовать все взаимосвязи и взаимозависимости между объектами социотехнической системы в виде нечетких отношений взаимовлияния, и, таким образом, применить для аппарат нечеткой каузальной алгебры.

Для решения задач идентификации кластеров в социотехнических системах обоснована совокупность показателей, основанная на нечетких отношениях взаимовлияния между этими объектами, и сформированная исходя из анализа нечетких когнитивных моделей.

Из всего множества системных показателей, основанных на согласованном нечетком отношении взаимовлияния для идентификации кластеров социотехнической системы по результатам проведенных исследований обоснован выбор совокупности следующих показателей: воздействия объекта на социотехническую систему; воздействия социотехнической системы на объект; согласованного влияния объекта на социотехническую систему.

Описана процедура кластеризации объектов и идентификация кластеров социотехнической системы, позволяющая в итоге оценить нечеткие степени принадлежности этих объектов к кластерам, а также определить их характеристические особенности по результатам анализа свойств (производственно-технологических, коммерческих, территориальных, логистических и иных) объектов, относящихся к каждому выделенному кластеру.

Для анализа устойчивости кластеров объектов в социотехнических системах предложен оригинальный способ, заключающийся в анализе результатов согласования нечетких отношений взаимовлияния между объектами каждого из идентифицированных кластеров социотехнической системы.

Представлены результаты использования предлагаемого подхода для идентификации и анализа устойчивости энергетической системы Смоленского региона.

Работа выполнена при финансовой поддержке РФФИ (проект № 18-29-03088) и Министерства науки и высшего образования РФ в рамках государственного задания № FSWF-2020-0019.

Список литературы

1. Savaget P., Geissdoerfer M., Kharrazi A. and Evansa S. The theoretical foundations of sociotechnical systems change for sustainability: A system-atic literature review // Journal of Cleaner Production. 2019. Vol. 206(1), pp. 878–892. DOI: https://doi.org/10.1016/j.jclepro.2018.09.208.
2. Borrás S. and Edler J. (Eds) The Governance of Socio-Technical Systems: Explaining Change. – Cheltenham: Edward Elgar Publishing. 2014. https://doi.org/10.4337/9781784710194.
3. Nemtinov V., Zazulya A., Kapustin V. and Nemtinova Y. 2019 Analysis of decision-making options in complex technical system design // Journal of Physics: Conference Series. Vol. 1278(1), p. 012018. https://doi.org/10.1088/1742-6596/1278/1/012018.
4. Ceschin F. and Gaziulusoy I. Evolution of design for sustainability: from product design to design for system innovations and transitions // Design Studies. 2016. Vol. 47, pp. 118–163. DOI: 10.1016/j.destud.2016.09.002.
5. Geels F.W. Ontologies, socio-technical transitions (to sustainability), and the multi-level perspective // Research Policy. 2010. Vol. 39, pp. 495–510. DOI: 10.1016/j.respol.2010.01.022.
6. Martin B R The evolution of science policy and innovation studies // Re-search Policy. 2012. Vol. 41, pp. 1219–1239. DOI: 10.1016/j.respol.2012.03.012.
7. Силов В.Б. Принятие стратегических решений в нечеткой обстановке. – М.: ИНПРО–РЕС, 1995 – 228 с.
8. Borisov V., Dli M., Zaenchkovsky A. and Fedulov Ya. Method for identifi-cation, stability analysis and the dynamics monitoring of sociotechnical clusters // Journal of Physics: Conference Series. 1553(2020), 012018 doi:10.1088/1742-6596/1553/1/012018.
9. Факторный, дискриминантный и кластерный анализ: Пер. с англ. / Дж.О. Ким, Ч.У. Мьюллер, У.Р. Клекка и др.; Под ред. И.С. Енюкова. – М.: Финансы и статистика, 1989. – 215 с.
10. Kaufman L., Rousseeuw P.J. Finding Groups in Data. An Introduction to Cluster Analysis. – John Wiley & Sons Inc., 2005. – PP. 83–102.
11. Bezdek J.C., Keller J., Krisnapuram R., Pal N. Fuzzy Models and Algo-rithms for Pattern Recognition and Image Processing. – Springer Science, New York, 2005. – 776 p.

References

1. Savaget P., Geissdoerfer M., Kharrazi A. and Evansa S. The theoretical foundations of sociotechnical systems change for sustainability: A system-atic literature review // Journal of Cleaner Production. 2019. Vol. 206(1), pp. 878–892. DOI: https://doi.org/10.1016/j.jclepro.2018.09.208.
2. Borrás S. and Edler J. (Eds) The Governance of Socio-Technical Systems: Explaining Change. – Cheltenham: Edward Elgar Publishing. 2014. https://doi.org/10.4337/9781784710194.
3. Nemtinov V., Zazulya A., Kapustin V. and Nemtinova Y. 2019 Analysis of decision-making options in complex technical system design // Journal of Physics: Conference Series. Vol. 1278(1), p. 012018. https://doi.org/10.1088/1742-6596/1278/1/012018.

4. Ceschin F. and Gaziulusoy I. Evolution of design for sustainability: from product design to design for system innovations and transitions // Design Studies. 2016. Vol. 47, pp. 118–163. DOI: 10.1016/j.destud.2016.09.002.
5. Geels F.W. Ontologies, socio-technical transitions (to sustainability), and the multi-level perspective // Research Policy. 2010. Vol. 39, pp. 495–510. DOI: 10.1016/j.respol.2010.01.022.
6. Martin B R The evolution of science policy and innovation studies // Re-search Policy. 2012. Vol. 41, pp. 1219–1239. DOI: 10.1016/j.respol.2012.03.012.
7. Silov V. Making strategic decisions in a fuzzy environment. - M .: INPRO-RES, 1995 - 228 p.
8. Borisov V., Dli M., Zaenchkovsky A. and Fedulov Ya. Method for identifi-cation, stability analysis and the dynamics monitoring of sociotechnical clusters // Journal of Physics: Conference Series. 1553(2020), 012018 doi:10.1088/1742-6596/1553/1/012018.
9. Факторный, дискриминантный и кластерный анализ: Пер. с англ. / Дж.О. Ким, Ч.У. Мьюллер, У.Р. Клекка и др.; Под ред. И.С. Енюкова. – М.: Финансы и статистика, 1989. – 215 с.
10. Kaufman L., Rousseeuw P.J. Finding Groups in Data. An Introduction to Cluster Analysis. – John Wiley & Sons Inc., 2005. – PP. 83–102.
11. Bezdek J.C., Keller J., Krisnapuram R., Pal N. Fuzzy Models and Algo-rithms for Pattern Recognition and Image Processing. – Springer Science, New York, 2005. – 776 p.