РЕФЕРАТ

Расчетно-пояснительная записка содержит 52 с., 11 рис., 3 табл., 18 ист., 4 прил.

Объектом работы является анализ активности пользователей системы автоматизированного проектирования.

Целью работы является разработка и программная реализация метода анализа активности пользователей системы автоматизированного проектирования с использованием поиска последовательных шаблонов

Ключевые слова: анализ активности пользователей, последовательные шаблоны, ассоциативные правила, система автоматизированного проектирования.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

ПО — программное обеспечение

САПР — система автоматизированного проектирования

ЧВП — часто встречающаяся последовательность

СУБД — система управления базами данных

ВВЕДЕНИЕ

Уровень удобства использования программного интерфейса влияет на качество всего ПО в целом. Признаком недостаточного уровня удобства использования является наличие проблем взаимодействия пользователя с пользовательским интерфейсом. Они могут быть связаны либо со сложностью формулирования плана действий (принятия решений, что делать дальше), либо с непониманием ответа системы (как изменения в интерфейсе связаны с выполненными действиями) [1].

Проблемы взаимодействия в большинстве случаев можно определить по наличию в данных активности пользователей определенных последовательностей действий (шаблонов). Для их обнаружения применяются различные методы анализа собираемых данных – как требующие ручного анализа (например, тепловые карты [2, 3]), так и использующие алгоритмы автоматического анализа [1] на основе шаблонов, выявленных исследователями ранее [4, 5, 6]. Автоматический анализ экономит время и деньги, так как эксперты вместо анализа всех данных фокусируют внимание на отдельных областях пользовательского интерфейса, где были выявлены соответствующие шаблоны.

Цель работы – разработать и программно реализовать метод анализа активности пользователей САПР с использованием поиска последовательных шаблонов.

Задачи работы:

— рассмотреть существующие решения в области анализа активности пользователей, выбрать для них критерии оценки и сравнить;

— формализовать задачу в виде IDEF0-диаграммы;

— разработать метод анализа активности пользователей САПР с использованием поиска последовательных шаблонов;

— разработать программное обеспечение, реализующее описанный метод;

— исследовать характеристики разработанного метода.

1 Аналитический раздел

1.1 Активность пользователей

Тестирование удобства использования программного обеспечения обычно состоит из двух этапов. Первый этап заключается в сборе данных о действиях, совершаемых пользователями посредством взаимодействия с графическим интерфейсом программы (движение курсора мыши, нажатие клавиш мыши, нажатие клавиш клавиатуры и т.д.), и характеристиках действий (координаты курсора, частота нажатия, используемые клавиши и т.д.). Такие данные обозначаются устоявшимся термином «активность пользователей». Второй этап – анализ этих данных экспертом с целью выявления проблем связанных с удобством использования, что является трудоемкой задачей. Поэтому, встает вопрос о хотя бы частичной автоматизации этого этапа, для чего требуется наличие соответствующих моделей и алгоритмов.

1.1.1 Шаблоны поведения пользователя

По мнению многих исследователей (например, авторов [1, 4, 5, 6]), индикатором проблем удобства использования может являться наличие часто повторяемых одинаковых последовательностей действий. Они могут означать, что пользователь пытается достичь цели и каждый раз терпит неудачу. Например, пользователь пытается взаимодействовать с изображением, которое он принял за кнопку [1], или пользователь нажимает кнопку и каждый раз получает ошибку.

В работе [4] выделен ряд шаблонов, связанных с выполнением пользователем поставленных задач, например, шаблон «Отмена действия», когда пользователь отменяет действие сразу после его выполнения, или шаблон «Повторение действий», когда пользователь часто повторяет простые действия (клики мыши или нажатие клавиш). Наличие второго шаблона может означать недостаточную отзывчивость интерфейса, которая ошибочно приводит пользователя к мысли, что система не распознает его действие.

Отдельные исследователи предлагают отслеживать более простые индикаторы: количество вызовов онлайн-справки, количество действий отмены, частое открытие-закрытие выпадающих списков, нажатие одной и той же кнопки более одного раза и т.д. [5]. Другие исследователи основываются на обнаружении проблем поиска информации пользователем в процессе просмотра веб-сайта [6]. Например, выделяется шаблон вертикального или горизонтального перемещения курсора мыши. В процессе визуального поиска на странице пользователь обычно перемещает курсор вслед за элементами, а значит, тратит много времени на поиск элемента.

Перечисленные методы поиска шаблонов поведения пользователей имеют много общего с задачей поиска последовательных шаблонов из области интеллектуального анализа данных [7]. В большинстве случаев все шаблоны являются последовательными, варьируются лишь анализируемые события. Однако данные активности пользователей почти всегда представляют собой не короткие транзакции, а большие наборы действий, которые в большинстве случаев невозможно корректно разделить на поднаборы [2, 3].

Поиск последовательных шаблонов давно и активно применяется в области торговли [8]. Поиск наиболее частых наборов позволяет получать информацию о том, через какой промежуток времени после покупки товара «А» человек наиболее склонен купить товар «Б» или в какой последовательности приобретаются товары. Получаемые закономерности в действиях покупателей можно использовать для персонализации клиентов, стимулирования продаж определенных товаров, управления запасами [8]. Это позволяет, с одной стороны, увеличить продажи, с другой – предложить клиентам товар, который, скорее всего, будет им интересен, а значит, минимизировать их временные затраты на поиск.

Как уже отмечалось, одной из возможных причин появления регулярно повторяющихся шаблонов в данных активности пользователей является наличие ошибок или затруднений при взаимодействии с интерфейсом. В этом случае может наблюдаться снижение и результативности, и эффективности пользователей. Следовательно, уменьшение числа подобных шаблонов снижает риск возникновения ошибок.

Другой возможной причиной наличия повторяющихся шаблонов в данных активности пользователей является потребность выполнения одних и тех же повторяющихся цепочек действий для выполнения поставленных задач. Закономерно, что автоматизация промежуточных действий уменьшает затраты ресурсов. Следовательно, чем меньше пользователь совершает однотипных цепочек действий, тем меньше он затрачивает ресурсов, а значит, тем эффективнее взаимодействие.

Конечно, при этом отмечается, что повторяющиеся шаблоны могут быть образованы из-за повторяющихся задач, которые либо невозможно или нецелесообразно автоматизировать, либо являются нормальным корректным поведением [1]. Поэтому требуется понимание семантики шаблонов и конкретных действий.

1.2 Существующие решения

1.2.1 Математическая модель пользовательской активности ПО

В статье [9] предлагается математическая модель активности пользователей ПО, основанная на теории последовательных шаблонов для предметной области оценки удобства использования. Данный способ совмещает анализ временных характеристик с вычислением уровней поддержки последовательных шаблонов.

Модель состоит из следующих элементов:

— множество событий с атрибутами;

— множество классов событий;

— функция классификации событий;

— множество сессий до классификации;

— множество сессий после классификации и фильтрации;

— множество последовательных шаблонов;

— множество значений поддержки последовательных шаблонов;

— функция преобразования класса событий в затрачиваемое время.

Данная модель может найти применение при оценке удобства использования пользовательских интерфейсов и для решения задач повышения эффективности взаимодействия пользователей с ПО.

Имея значения поддержки и затрачиваемого времени для каждого шаблона, эксперт может сконцентрироваться на наиболее значимых из них для процесса работы пользователей с ПО в целом. Набор шаблонов при этом будет зависеть от целей проводимого анализа.

Далее эксперт может выдвинуть гипотезы о необходимых изменениях в пользовательском интерфейсе для повышения эффективности взаимодействия пользователей с ПО. При принятии решений эксперту необходимо учитывать множество различных факторов: особенности ПО, психологические факторы использования ПО и особенности пользователей.

Изменение пользовательского интерфейса повлечет изменение множеств событий, сессий и последовательных шаблонов, так как изменится последовательность действий, необходимых для достижения пользователями поставленных целей.

Таким образом, можно утверждать, что задачей эксперта становится переход от текущей модели активности пользователей к новой, с иным составом сессий и шаблонов, следовательно, и иными значениями поддержки шаблонов и затратами времени пользователей.

После внесения изменений в программный интерфейс возможны повторный сбор и анализ данных активности пользователей, что может подтвердить либо опровергнуть выдвинутую ранее гипотезу.

Недостатком данной модели является использование заранее предопределенных шаблонов.

1.2.2 Алгоритм получения ассоциативных правил Apriori

Базовым алгоритмом, применяемым для получения ассоциативных правил, является алгоритм Apriori [10], автором которого является Ракеш Агравал (Rakesh Agrawal). Алгоритм Apriori использует стратегию поиска в ширину и осуществляет его снизу-вверх, последовательно перебирая кандидатов. В алгоритме используются две структуры данных: C_i — для хранения множества кандидатов в частые множества признаков длины i и F_i — для хранения частых множеств признаков длины i. Каждая структура имеет два поля — itemset, сохраняющее множество признаков, и support, которое хранит величину поддержки этого множества признаков. Ниже представлена формальная запись алгоритма состоящего из двух частей: самого Apriori и вспомогательной процедуры AprioriGen.

Apriori($Context, min\_supp$). Context - набор данных, min_supp - минимальная поддержка, I_F – все частые множества признаков.

$C_1 \leftarrow \{1 - itemsets\}$
$i \leftarrow 1$
while$(C_i \neq 0)$
do $\begin{cases} SupportCount(C_i) \\ F_i \leftarrow \{f \in C_i | f.support \geq min\_supp\}//\text{F – частые множества признаков} \\ C_{i+1} \leftarrow AprioriGen(F_i)//\text{C – кандидаты} \\ i++ \end{cases}$
$I_F \leftarrow \cup F_i$
return(I_F)

AprioriGen(F_i). F_i - частые множества признаков длины i, C_{i+1} – потенциальные кандидаты частых множеств признаков.

$C_{i+1} \leftarrow \{p \cup q | p, q \in F_i, |p \cap q| = k - 2\}$
for each $c \in C_{i+1}$ // удаление
do $\begin{cases} S \leftarrow (i-1) - \text{элементарные подмножества } c \\ \textbf{for each } s \in S \text{// удаление} \\ \textbf{do} \begin{cases} \textbf{if } (s \notin F_i) \\ \textbf{then } C_{i+1} \leftarrow C_{i+1} \backslash c \end{cases} \\ \textbf{return}(C_{i+1}) \end{cases}$

Основной особенностью алгоритма можно считать использование свойства антимонотонности, которое гласит, что поддержка любого набора элементов не может превышать минимальной поддержки любого из его подмножеств. Именно благодаря этому свойству перебор не является «жадным» и позволяет обрабатывать большие массивы информации за секунды.

Существуют различные модификации алгоритма Apriori и иные алгоритмы [11], значительно оптимизированные под определенные ситуации. Можно утверждать, что применение существующего проработанного аппарата теории последовательных шаблонов позволит реализовать поиск неизвестных ранее шаблонов взаимодействия пользователей с интерфейсом программы.

Алгоритмическая сложность данного метода экспоненциальна: $O(|D| \cdot |I| \cdot 2^{|I|})$, где $|D|$ – количество транзакций в исходном наборе данных, а $|I|$ – общее число предметов [12]. Но это верно только для худшего случая, если задать минимальный уровень поддержки = 0, что не имеет практического смысла, т.к. интерес представляют ассоциативные правила с высоким уровнем поддержки.

1.2.3 Алгоритм GSP

Алгоритм GSP (англ. Generalized Sequential Pattern, обобщенный секвенциальный паттерн) является модификацией алгоритма AprioriAll, учитывающей ограничения по времени между соседними транзакциями [13, 14].

В случае с алгоритмом GSP требуется учитывать дополнительные условия, чтобы определить, содержит ли последовательность указанную подпоследовательность [15].

Введем такие параметры, как минимальное и максимальное допустимое время между транзакциями (min_gap и max_gap), а также понятие скользящего окна, размера win_size. Допускается, что элемент последовательности может состоять не из одной, а из нескольких транзакций, если разница во времени между ними меньше, чем размер окна.

Последовательность $d = \langle d_1 ... d_m \rangle$ содержит последовательность $s = \langle s_1 ... s_m \rangle$, если существуют такие целые числа $l_1 \leqslant u_1 < l_2 \leqslant u_2 < ... < l_n \leqslant u_n$, что:

— s_i содержится в объединении d_k, где $l_i \leqslant k \leqslant u_i, 1 \leqslant i \leqslant n$.

— $t_{\text{транзакции}}(d_{l[i]}) - t_{\text{транзакции}}(d_{u[i-1]}) \leqslant win\_size, 1 \leqslant i \leqslant n$.

— $min\_gap \leqslant t_{\text{транзакции}}(d_{l[i]}) - t_{\text{транзакции}}(d_{u[i-1]}) \leqslant max\_gap$,
$2 \leqslant i \leqslant n$.

Выполнение алгоритма GSP предусматривает несколько проходов по исходному набору данных. При первом проходе вычисляется поддержка для каждого предмета и из них выделяются частые. Каждый подобный предмет представляет собой одноэлементную последовательность. В начале каждого последующего прохода имеется некоторое число ЧВП, выявленных на предыдущем шаге алгоритма. Из них будут формироваться более длинные последовательности-кандидаты.

Каждый кандидат представляет собой последовательность, длина которой *на один больше* чем у последовательностей, из которых кандидат был сформирован. Таким образом, число элементов всех кандидатов одинаково. После формирования кандидатов происходит вычисление их поддержки. В конце шага определяется, какие кандидаты являются ЧВП. Найденные ЧВП послужат исходными данными для следующего шага алгоритма. Работа алгоритма завершается тогда, когда не найдено ни одной новой ЧВП в конце очередного шага, или когда невозможно сформировать новых кандидатов.

Таким образом, в работе алгоритма можно выделить следующие основные этапы:

1. Генерация кандидатов.
    1. Объединение.
    2. Упрощение.
2. Подсчет поддержки кандидатов.

Рассмотрим эти операции более подробно.

Этап 1. Генерация кандидатов. Пусть L_k содержит все частые k-последовательности, а C_k – множество кандидатов из k-последовательностей. В начале каждого шага имеем L_{k-1} – набор из $(k-1)$ ЧВП. На их основе необходимо построить набор всех k ЧВП.

Введем понятие *смежной подпоследовательности.*

При наличии последовательности $s = \langle s1, s2..., sn\rangle$ и подпоследовательности c, c будет являться *смежной последовательностью* s, если соблюдается одно из условий:

— c получается из s при удалении предмета из первого $\{s_1\}$ или последнего $\{s_n\}$ предметного множества;

— c получается из s при удалении одного предмета из предметного множества s_i, если в его составе не менее двух предметов;

— c – смежная подпоследовательность c', где c' – смежная подпоследовательность s.

Например, дана последовательность $s = \langle\{1,2\},\{3,4\},\{5\},\{6\}\rangle$. Последовательности $\langle\{2\},\{3,4\},\{5\}\rangle$, $\langle\{1,2\},\{3\},\{5\},\{6\}\rangle$ и $\langle\{3\},\{5\}\rangle$ являются смежными подпоследовательностями s, а последовательности $\langle\{1,2\},\{3,4\},\{6\}\rangle$ и $\langle\{1\},\{5\},\{6\}\rangle$ таковыми не являются.

Если некоторая последовательность содержит последовательность s, то она также содержит и все смежные подпоследовательности s.

Генерация кандидатов происходит в два этапа.

Этап 1.1. Объединение (англ. *join*). Создаем последовательности-кандидаты путем объединения двух последовательностей L_{k-1} и L_{k-1}. Последовательность s_1 объединяется с s_2, если подпоследовательность, образующаяся путем удаления первого предмета из s_1, будет та же, что и в случае удаления последнего предмета из s_2. Объединение последовательностей происходит путем добавления к s_1 соответствующего предмета из последнего предметного множества s_2. Причем возможны два варианта:

— если последний предмет из s_2 составлял одноэлементное предметное множество, то при объединении он будет добавлен к s_1 как новое предметное множество;

— в противном случае, он будет включен в последнее предметное множество s_1 как его элемент.

При объединении L_1 с L_1 нужно добавить предмет к s_2 как отдельное предметное множество, а также в качестве дополнительного предмета в предметное множество последовательности s_1. Так, объединение $\langle\{x\}\rangle$ с $\langle\{y\}\rangle$ дадут как $\langle\{x, y\}\rangle$, так и $\langle\{x\},\{y\}\rangle$. Причем x и y упорядочены.

Этап 1.2. Упрощение (англ. *prune*). Удаляем последовательности кандидаты, которые содержат смежные $(k-1)$-последовательности, чья поддержка меньше минимально допустимой.

Этап 2. Подсчет поддержки кандидатов. Сканируя набор последовательностей, обрабатываем их по очереди. Для тех кандидатов, которые содержатся в обрабатываемой последовательности, увеличиваем значение поддержки на единицу.

Алгоритмическая сложность данного метода экспоненциальна: $O(|I|^l)$, где $|I|$ – общее число предметов, а l – длина наибольшей часто встречающейся последовательности [12].

1.2.4 Метод оценки эффективности интерфейса GOMS

Метод GOMS (сокращение от Goals, Operators, Methods and Selection Rules – Цели, Операторы, Методы и Правила выбора) – это семейство мето-

дов, позволяющих провести моделирование выполнения той или иной задачи пользователем и на основе такой модели оценить качество интерфейса.

Идея метода заключается в разбиении взаимодействия пользователя с интерфейсом на атомарные физические и когнитивные действия. Обладая знаниями о метриках каждой из таких составляющих, можно делать заключение об эффективности взаимодействия в целом: оценка эффективности интерфейса сводится к разбиению типовых задач на элементарные действия и сложению метрик каждого из них.

Метод GOMS включает в себя модель Keystroke-level Model (KLM) [16], которая выделяет следующие элементарные задачи и длительность каждой из них (рассчитанные на основе усредненных данных лабораторных испытаний):

— K – нажатие на клавишу в зависимости от уровня владения клавиатурой: профессиональный наборщик – 0.08 сек., эксперт – 0.12 сек., частая работа с текстом – 0.20 сек., продвинутый пользователь – 0.28 сек., неуверенный пользователь – 0.5 сек., не знакомый с клавиатурой – 1.2 сек.;

— P – указание курсором мыши на объект – 1.1 сек.;

— B – нажатие или отпускание мыши – 0.1 сек.;

— M – умственная подготовка, выбор действия – 1.2 сек.;

— H – перемещение руки в исходное положение на клавиатуре – 0.4 сек;

— R – ожидание ответа системы, зависящее от времени выполнения системой запрошенной операции.

Оценка времени на решение задачи сводится к сложению продолжительностей каждой из простейших составляющих. Например, задача, состоящая из классов (P, P, B), потребует для завершения 2.3 сек. (1.1 сек. + 1.1 сек. + 0.1 сек.).

1.3 Сравнение

В таблице 1.1 приведено сравнение рассмотренных методов.

Таблица 1.1 — Сравнение рассмотренных методов

Метод	Требование к входным данным	Учет времени транз-ий	Сложность алгоритма
Мат. модель пользов. актив. ПО	Множество событий, функция классификации событий, множество сессий, множество последовательных шаблонов	Нет	$O(n \cdot m)$, где n – кол-во шаблонов, m – кол-во сессий
Apriori	Транзакции с набором элементов и минимальный уровень поддержки	Нет	$O(\|D\| \cdot \|I\| \cdot 2^{\|I\|})$, где $\|D\|$ – кол-во транзакций, $\|I\|$ – общее число предметов
GSP	База данных с полями: id последовательности, id и время транзакции, набор элементов и минимальный уровень поддержки	Да	$O(\|I\|^l)$, где $\|I\|$ – общее число предметов, l – длина наибольшей ЧВП
GOMS	Последовательность действий	Нет	$O(n)$, где n – число действий в послед-ти

Вычисление уровней поддержки шаблонов поведения пользователя позволяет ранжировать их по степени приоритета для детального анализа. Методы поиска ассоциативных правил и последовательных шаблонов позволяют найти новые (неизвестные ранее) шаблоны взаимодействия пользователей с программным обеспечением. А анализ временных характеристик, позволяет оценить эффективность взаимодействия пользователя с интерфейсом.

1.4 Формализованная постановка задачи

Поскольку активность пользователей представляет собой последовательность действий и их характеристик, производящихся в определенный момент времени, то за основу был выбран алгоритм GSP, т.к. он учитывает время совершения транзакций.

Для выполнения дальнейшей работы необходимо формализовать задачу анализа активности пользователей САПР. Поставленная задача представлена в нотации IDEF0 на рисунке 1.1. На вход программе подаются информация о выполненных командах и пользовательские параметры: минимальный уровень поддержки, минимальный и максимальный разрывы между командами. Используя методы поиска последовательных шаблонов система определяет часто встречающиеся последовательности команд.

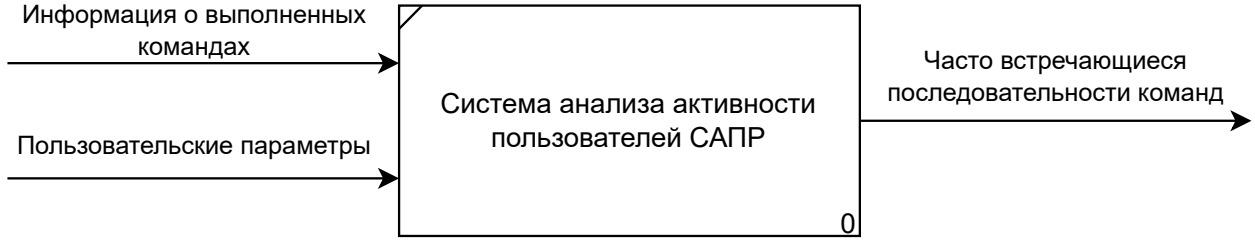

Рисунок 1.1 — IDEF0-диаграмма нулевого уровня

Вывод из аналитического раздела

В данном разделе был проведен обзор существующих методов анализа пользовательской активности, сформулированы критерии для их оценки, проведено сравнение рассмотренных методов и формализована задача в виде IDEF0-диаграммы.

2 Конструкторский раздел

В случае анализа активности пользователей САПР, зачастую, данные представлены в виде последовательности команд с параметрами. За одну транзакцию примем выполнение одной команды. Кроме этого, необходимо отмечать к какой сессии принадлежит каждая команда. В таком случае поддержкой последовательности команд будет отношение числа сессий поддерживающих данную последовательность к общему их количеству.

2.1 Особенности предлагаемого метода

Обычно логи представлены в текстовом виде, поэтому перед обработкой их алгоритмом, данные будут преобразовываться в таблицу базы данных со следующими полями:

— id;

— id сессии в течение которой была исполнена команда;

— время в которое команда была выполнена;

— имя команды.

В алгоритме GSP элемент последовательности может содержать несколько транзакций, если они были совершены в пределах заданного скользящего окна. Кроме этого, в одну транзакцию может входить несколько предметов. В таком случае не учитывается в каком порядке были выполнены операции в пределах одной транзакции. Данный подход пригоден для использования в области торговли, но в случае анализа активности пользователей, все действия выполняются в определенной последовательности и учитывание порядка каждого из них даст больше информации в результате. Поэтому для разрабатываемого метода элемент последовательности может состоять только из одной команды. Следовательно, нет необходимости в использовании скользящего окна в пределах которого, совершенные транзакции считаются совершенными одновременно. Но в таком случае возникает проблема с определением порядка выполнения команд выполненных в один момент времени.

Чтобы ее решить, будем считать, что команды выполнились в том порядке, в котором они были записаны в лог.

2.2 Ключевые этапы алгоритма

Как и в алгоритме GSP, разрабатываемый метод будет состоять из двух основных этапов:

1. Генерация кандидатов.
2. Подсчет поддержки кандидатов.

Генерация кандидатов. Данный этап описывает как генерируются последовательности-кандидаты перед проверкой их уровня поддержки. Нужно сгенерировать всех возможных кандидатов, стараясь свести их количество к минимуму.

Как было описано выше, в алгоритме GSP данный этап состоит из двух частей: объединение и упрощение. Но поскольку в нашем случае каждый элемент последовательности может состоять только из одной команды, отпадает необходимость в упрощении. Также, стоит отметить, что на первом шаге алгоритма в качестве кандидатов нужно просто взять всевозможные одноэлементные последовательности.

На рисунке 2.1 представлена блок-схема данного этапа.

Рисунок 2.1 — Генерация кандидатов

Подсчет поддержки кандидатов. После получения списка кандидатов, нужно определить какие последовательности удовлетворяют заданному минимальному уровню поддержки, а какие нет. Для этого необходимо для каждого кандидата определить кол-во сессий поддерживающих его. На рисунках 2.2-2.4 приведена схема проверки поддержки кандидата сессией.

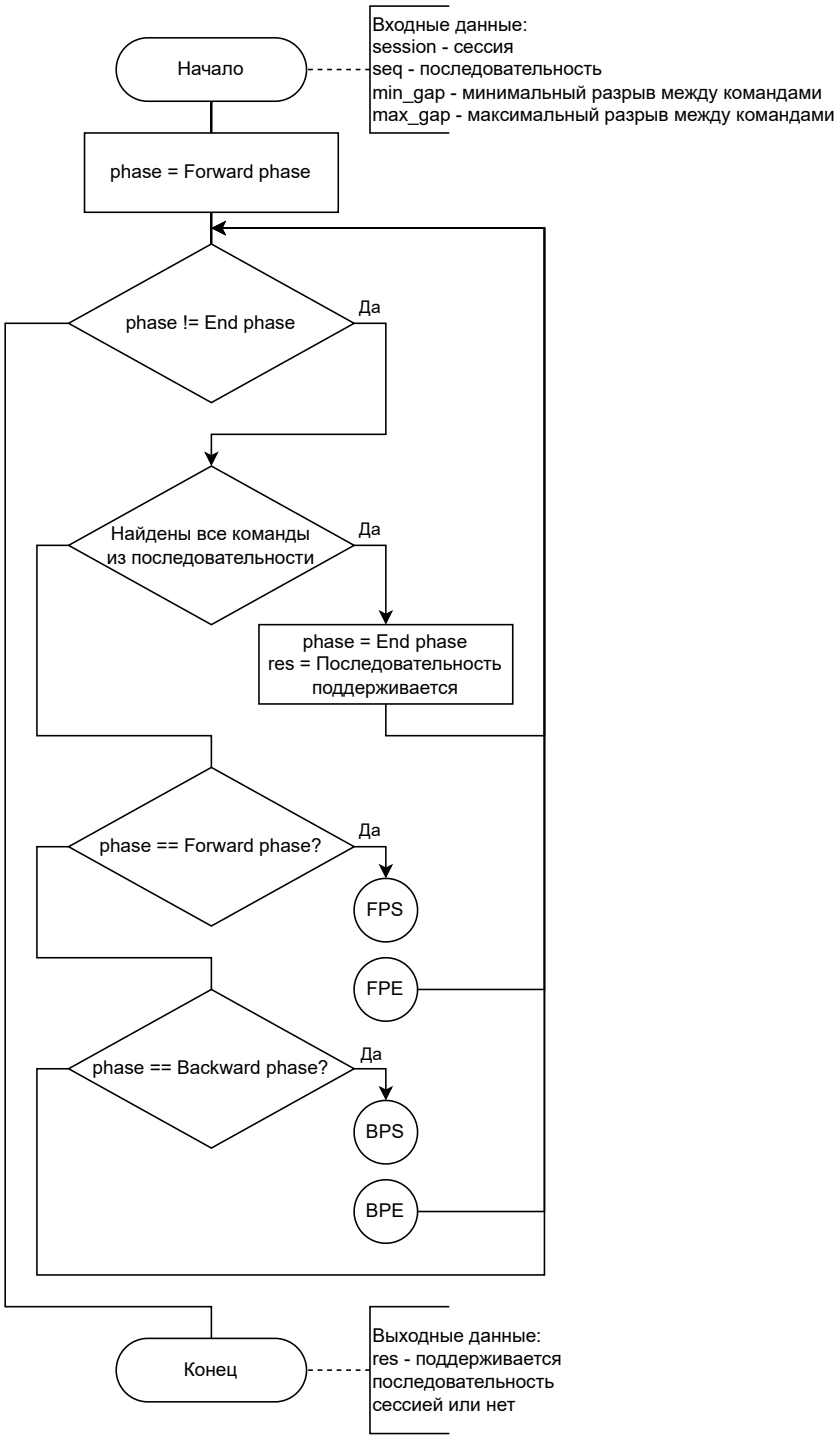

Рисунок 2.2 — Проверка поддержки кандидата сессией, часть 1

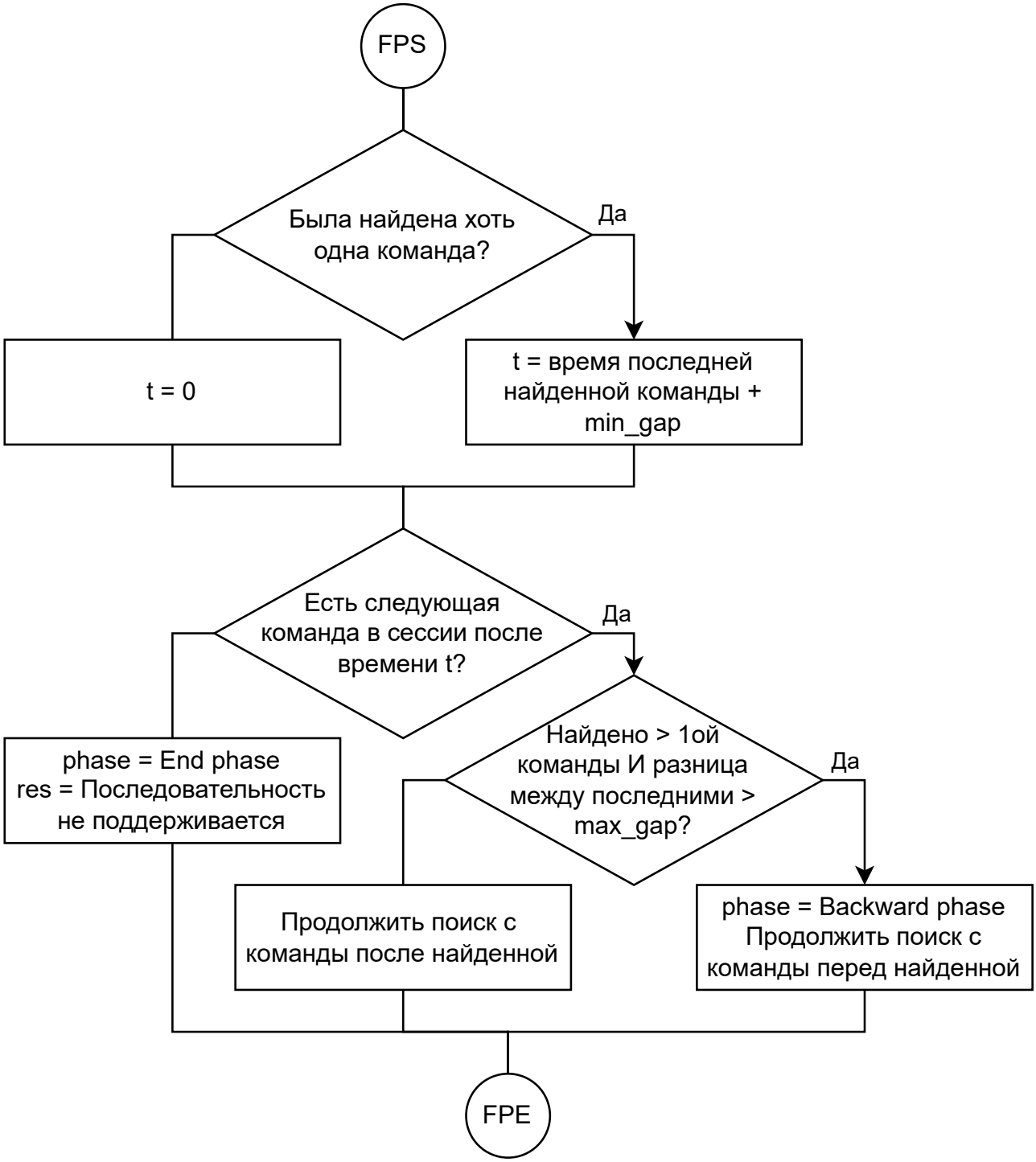

Рисунок 2.3 — Проверка поддержки кандидата сессией, часть 2

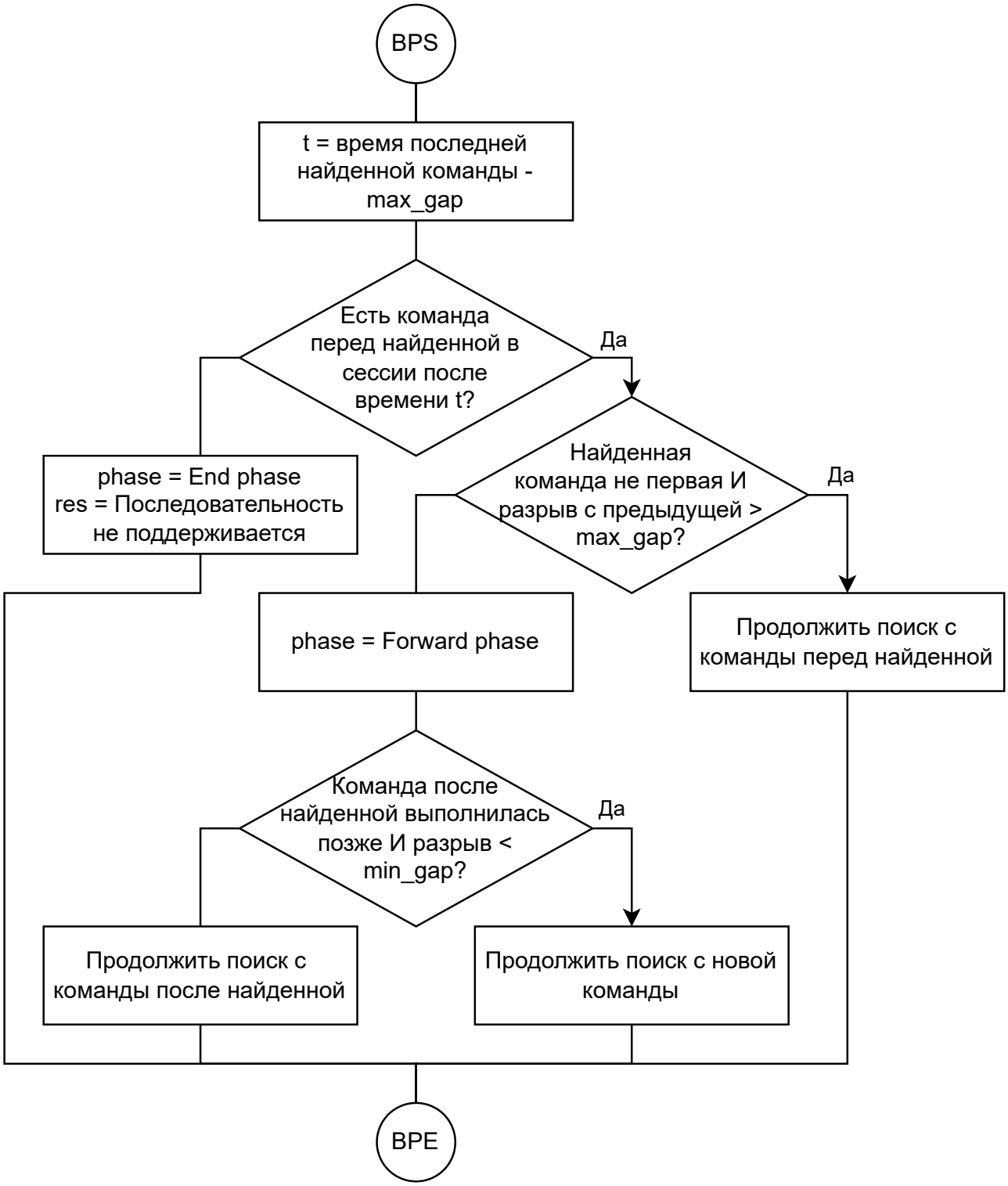

Рисунок 2.4 — Проверка поддержки кандидата сессией, часть 3

2.3 Используемые структуры данных

Для работы алгоритма необходимо описать структуры данных для следующих основных объектов:

— команда;

— последовательность;

— сессия.

Для команды, в соответствии с таблицей базы данных подающейся на вход алгоритму, выделены следующие параметры:

— id;

— id сессии в течение которой была исполнена команда;

— время в которое команда была выполнена;

— имя команды.

Последовательность будет хранить: массив команд и собственное значение поддержки.

Для сессии можно выделить следующие поля: id, массив команд, альтернативное представление сессии в виде массива односвязных списков хранящих время выполнения команд. Последнее поле сессии используется для подсчета поддержки кандидатов.

Таблица 2.1 — Пример сессии

Время транзакции	Команда
1	2
1	2
5	2
8	3
8	4

Таблица 2.2 — Альтернативное представление сессии

Команда	Времена
1	$\rightarrow$ NULL
2	$\rightarrow 1 \rightarrow 1 \rightarrow 5 \rightarrow$ NULL
3	$\rightarrow 8 \rightarrow$ NULL
4	$\rightarrow 8 \rightarrow$ NULL

Пример проверки поддержки последовательности сессией. Допустим есть сессия (таблицы 2.1-2.2) и нам нужно определить поддерживает ли она последовательность $\langle 2,3,4 \rangle$. Если задать минимальный и максимальный разрывы между командами 0 и 3 соответственно, то сначала будет найдена

команда 2 в момент времени 1. После этого, найдя команду 3 в момент времени 8, получится что разрыв между командами 2 и 3 больше максимального ($8 - 1 > 3$). Поэтому надо искать команду 2 начиная с времени $8 - 3 = 5$. Таким образом мы находим команду 2 в момент времени 5, и поскольку $5 < 8$, то найденная до этого команда 3 нас устраивает и поиск продолжается для последней команды 4 с момента времени 8. Находим искомую команду в момент времени 8 и получается что данная сессия поддерживает последовательность $\langle 2,3,4 \rangle$.

Если бы минимальный разрыв между командам был > 0, то последовательность $\langle 2,3,4 \rangle$ не поддерживалась бы этой сессией, т.к. разрыв между командами 3 и 4 равен нулю. А если задать максимальный разрыв между командами < 3, то разрыв между командами 2 и 3 не будет удовлетворять заданному условию и соответственно последовательность также не будет поддерживаться сессией.

Вывод из конструкторского раздела

В данном разделе был разработан метод анализа активности пользователей САПР с использованием поиска последовательных шаблонов, рассмотрены особенности предлагаемого метода, выделены и представлены в виде схем алгоритмов его основные этапы, а также описаны используемые структуры данных.

3 Технологический раздел

3.1 Основные инструменты, используемые для реализации

Язык программирования C++.

C++ – язык программирования общего назначения с уклоном в сторону системного программирования [17]

Данный язык, достаточно популярен и широко распространен, кроме этого, он имеет ряд плюсов, описанных ниже:

— высокая производительность: язык спроектирован так, чтобы дать программисту максимальный контроль над всеми аспектами структуры и порядка исполнения программы; один из базовых принципов C++ «не платишь за то, что не используешь» то есть ни одна из языковых возможностей, приводящая к дополнительным накладным расходам, не является обязательной для использования; имеется возможность работы с памятью на низком уровне;

— кроссплатформенность: стандарт языка C++ накладывает минимальные требования на ЭВМ для запуска скомпилированных программ;

— поддержка различных стилей программирования: традиционное императивное программирование (структурное, объектно-ориентированное), обобщенное программирование, функциональное программирование, порождающее метапрограммирование.

Исходя из вышеперечисленных плюсов, очевиден выбор данного языка программирования для реализации поставленной задачи.

Кроссплатформенный фреймворк Qt.

Qt – кроссплатформенный фреймворк для разработки программного обеспечения на языке программирования C++. Есть также «привязки» ко многим другим языкам программирования: Python — PyQt, PySide; Ruby — QtRuby; Java — Qt Jambi; PHP — PHP-Qt и другие. Поддерживаемые платформы включают Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, ОС Sailfish и другие [18].

Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путем простой компиляции программы для каждой системы без изменения исходного кода (кроссплатформенность). Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Является полностью объектно-ориентированным, расширяемым и поддерживающим технику компонентного программирования.

Комплектуется визуальной средой разработки графического интерфейса Qt Designer, позволяющей создавать диалоги и формы.

Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

Мета-объектная система — часть ядра фреймворка для поддержки в C++ таких возможностей, как сигналы и слоты для коммуникации между объектами в режиме реального времени и динамических свойств системы.

Одним из преимуществ проекта Qt является наличие качественной документации. Статьи документации снабжены большим количеством примеров. Исходный код самой библиотеки хорошо форматирован, подробно комментирован, что также упрощает изучение Qt.

Отличительная особенность — использование мета-объектного компилятора — предварительной системы обработки исходного кода. Расширение возможностей обеспечивается системой плагинов, которые возможно размещать непосредственно в панели визуального редактора. Но минусом получается то, что код написанный с помощью Qt нельзя скомпилировать на другом компьютере без установки фреймворка.

Для реализации проекта был выбран фреймворк Qt, т.к. он кроссплатформенный, упрощает создание интерфейса и взаимодействие с базами данных.

Среда разработки Qt creator.

Qt Creator (ранее известная под кодовым названием Greenhouse) — кроссплатформенная свободная IDE для языков C, C++ и QML. Разработана Trolltech (Digia) для работы с фреймворком Qt. Включает в себя графический интерфейс отладчика и визуальные средства разработки интерфейса как с использованием QtWidgets, так и QML. Поддерживаемые компиляторы: GCC, Clang, MinGW, MSVC, Linux ICC, GCCE, RVCT, WINSCW.

Основная задача Qt Creator — упростить разработку приложения с помощью фреймворка Qt на разных платформах. Поэтому для работы с данной библиотекой был выбран именно он.

Система версионного контроля git.

Для хранения исходников используется система Git (на портале github.com), т.к. это крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

СУБД SQLite

Для работы с базой данных была выбрана СУБД SQLite, т.к. она не работает в режиме сервера-клиента, а встраивается непосредственно в приложение. Это означает, что вся база данных хранится в одном файле, который обрабатывается непосредственно приложением.

3.2 Входные и выходные данные

Данная программа разрабатывается для анализа логов САПР NanoCAD, которые имеют следующий вид: каждая строчка, содержащая действие, начинается с даты и времени выполнения. Начало каждой команды обозначается ее названием, заключенной в символы '<' и '>'. Завершение команды обозначается аналогично, но перед названием команды добавляется символ '/'. Пример входного файла, см. в приложении А.

Также поддерживается считывание обезличенных логов, которые не содержат информации о параметрах команд, включая время выполнения. В таком случае время выполнения для каждой команды в сессии будет выстав-

лено автоматически, начиная с 0, увеличивая это значения на 1 для каждой следующей команды.

Кроме этого на вход программе подаются минимальный уровень поддержки, а также минимальный и максимальный разрывы между командами.

На выходе программа выдает часто встречающиеся последовательности команд, их уровни поддержки и коэффициент зависимости. Коэффициент зависимости показывает насколько команды в последовательности зависят друг от друга и считается как отношение поддержки последовательности к произведению поддержек всех подпоследовательностей состоящих из 1 команды. Если значение коэффициента $<= 1$, значит зависимости нету. Если же > 1, то зависимость есть. Чем больше единицы, тем вероятней то, что эти команды использовались вместе.

3.3 Реализация

За преобразование логов из текста в таблицу базы данных отвечает класс *LogReader*. Он может считать все файлы с расширением *.log в выбранной директории и её поддиректориях, записывая все команды в таблицу logs для выбранной на текущий момент базы данных. Также можно указать, нужно ли учитывать завершение команды как отдельное действие, если она началась и закончилась одновременно.

За взаимодействие с базами данных отвечает класс *DataBase*. Данный класс позволяет создавать базы данных, переключаться между ними и как записывать в них необходимые данные, так и считывать их.

За реализацию разрабатываемого метода отвечает класс *Calculator*. Он хранит входные параметры метода, а также необходимые для работы данные и последний полученный результат.

Описание данных классов приведено в приложении Б.

Пользовательский интерфейс состоит из 4 окон. *MainWindow* – основное окно в котором можно выбирать и просматривать базу данных, считывать в нее логи из выбранной директории, настраивать режимы считывания, задавать параметры метода, запускать его для текущей базы данных и наблюдать результат. *DataBaseWindow* и *ResWindow* используются для просмотра базы данных и результатов работы программы в отдельных окнах. Последнее окно *CmdListWindow* содержит расширенные настройки, в котором можно указать команды которые будут игнорироваться и которые означают начало новой сессии.

Интерфейс программы см. в приложении В.

3.4 Примеры работы программы

Для простоты и наглядности анализировались логи, где большинство сессий подряд повторялись одинаковые действия. На рисунке 3.1 приведен пример работы программы без учета последовательностей с повторяющимися по две или более подряд командами. На рисунке 3.2 показан результат анализа тех же логов, но с учетом последовательностей с повторяющимися командами.

Результат алгоритма

	Частая последовательность	support	lift
1	<Arc, Circle, EditUndo>	0.636	1.729
2	<Line, Arc, Circle, EditUndo>	0.636	1.729
3	<Arc, Circle>	0.636	1.571
4	<Line, Arc, Circle>	0.636	1.571
5	<Arc, EditUndo>	0.636	1.100
6	<Circle, EditUndo>	0.636	1.100
7	<Line, Arc, EditUndo>	0.636	1.100
8	<Line, Circle, EditUndo>	0.636	1.100
9	<Line, EditUndo>	0.909	1.000
10	<Line, Arc>	0.636	1.000
11	<Line, Circle>	0.636	1.000
12	<Line>	1.000	null
13	<EditUndo>	0.909	null
14	<Arc>	0.636	null
15	<Circle>	0.636	null

Рисунок 3.1 — Пример работы программы 2

	Частая последовательность	support	lift
1	<Line, Arc, Circle, EditUndo, EditUndo, EditUndo>	0.636	2.092
2	<Arc, Circle, EditUndo, EditUndo, EditUndo>	0.636	2.092
3	<Line, Arc, Circle, EditUndo, EditUndo>	0.636	1.901
4	<Arc, Circle, EditUndo, EditUndo>	0.636	1.901
5	<Line, Arc, Circle, EditUndo>	0.636	1.729
6	<Arc, Circle, EditUndo>	0.636	1.729
7	<Line, Arc, Circle>	0.636	1.571
8	<Arc, Circle>	0.636	1.571
9	<Line, Circle, EditUndo, EditUndo, EditUndo>	0.636	1.331
10	<Line, Arc, EditUndo, EditUndo, EditUndo>	0.636	1.331
11	<Arc, EditUndo, EditUndo, EditUndo>	0.636	1.331
12	<Circle, EditUndo, EditUndo, EditUndo>	0.636	1.331
13	<Arc, EditUndo, EditUndo>	0.636	1.210
14	<Circle, EditUndo, EditUndo>	0.636	1.210
15	<Line, Circle, EditUndo, EditUndo>	0.636	1.210
16	<Line, Arc, EditUndo, EditUndo>	0.636	1.210
17	<Circle, EditUndo>	0.636	1.100
18	<Line, Arc, EditUndo>	0.636	1.100
19	<Line, Circle, EditUndo>	0.636	1.100
20	<Arc, EditUndo>	0.636	1.100
21	<Line, EditUndo>	0.909	1.000
22	<Line, Arc>	0.636	1.000
23	<Line, Circle>	0.636	1.000
24	<Line, EditUndo, EditUndo>	0.818	0.990
25	<EditUndo, EditUndo>	0.818	0.990
26	<EditUndo, EditUndo, EditUndo>	0.727	0.968
27	<Line, EditUndo, EditUndo, EditUndo>	0.727	0.968
28	<Line>	1.000	null
29	<EditUndo>	0.909	null
30	<Arc>	0.636	null
31	<Circle>	0.636	null

Рисунок 3.2 — Пример работы программы 2

Вывод из технологического раздела

В данном разделе был обоснован выбор основных инструментов, используемых для реализации, описан формат входных и выходных данных, разработано программное обеспечение, реализующее описанный метод и приведены примеры работы программы.

4 Исследовательский раздел

Ниже приведены технические характеристики устройства, на котором были проведены эксперименты при помощи разработанного ПО:

— операционная система: Windows 10 (64-разрядная);

— оперативная память: 32 GB;

— процессор: Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz;

— количество ядер: 4;

— количество потоков: 8.

4.1 Сравнительный анализ времени выполнения метода в зависимости от параметров

Чтобы провести сравнительный анализ времени выполнения метода, замерялось время выполнения метода с разными значениями параметров и количеством записей 1000 раз, а затем делилось на количество замеров. При изменении минимального уровня поддержки параметр min_gap был равен нулю, а max_gap имел максимально возможное значение (2147483647). При изменении минимального разрыва между командами параметр min_sup был равен 0.01, а max_gap также имел максимально возможное значение. При изменении максимального разрыва между командами параметр min_sup был равен 0.01, а min_gap нулю. На рисунках 4.1-4.3 представлены результаты исследования в виде графиков.

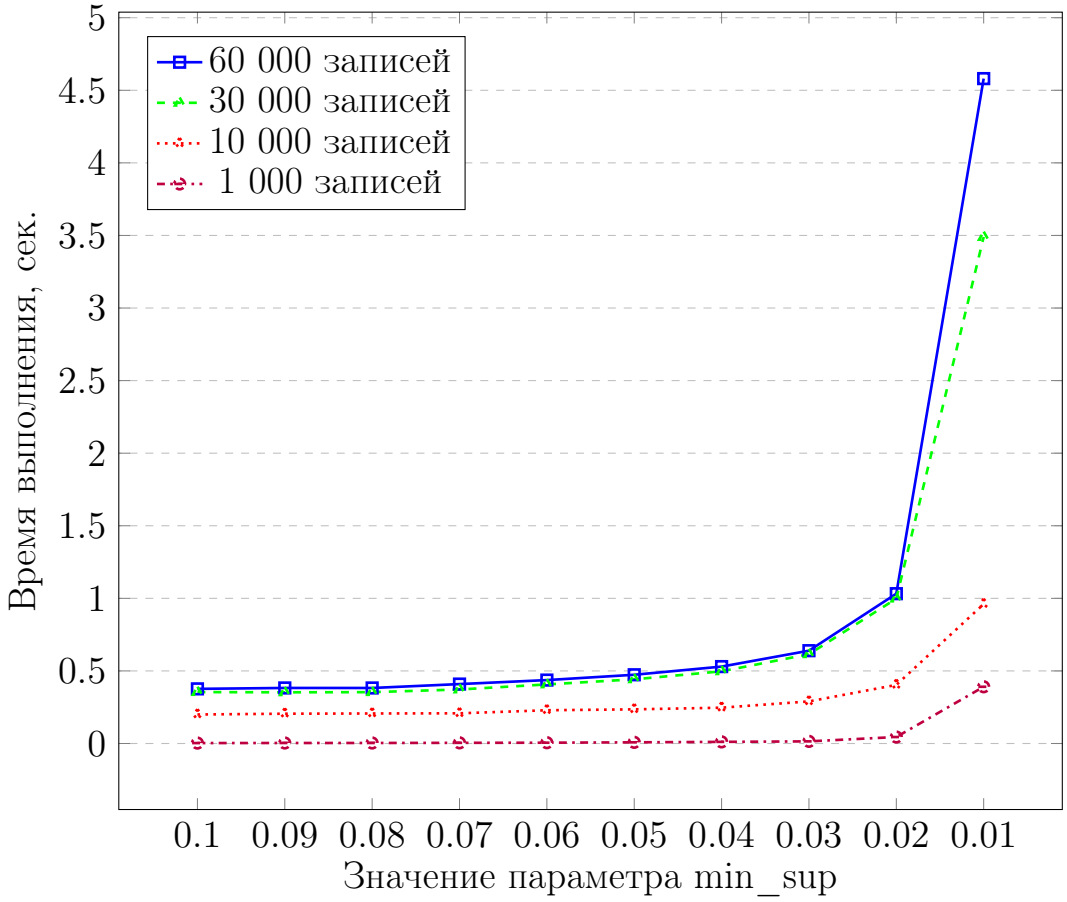

Рисунок 4.1 — Зависимость времени выполнения метода от минимальной поддержки

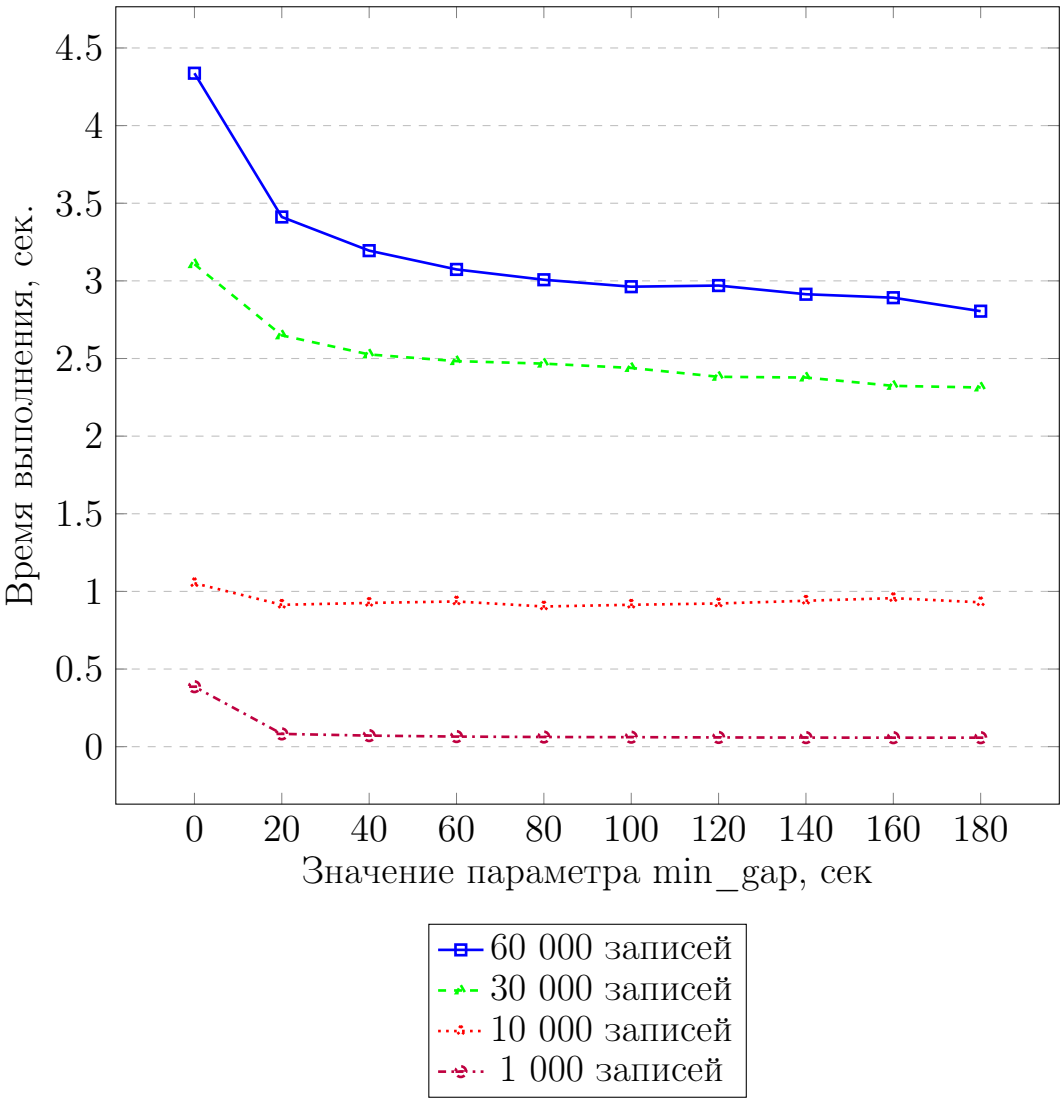

Рисунок 4.2 — Зависимость времени выполнения метода от минимального разрыва

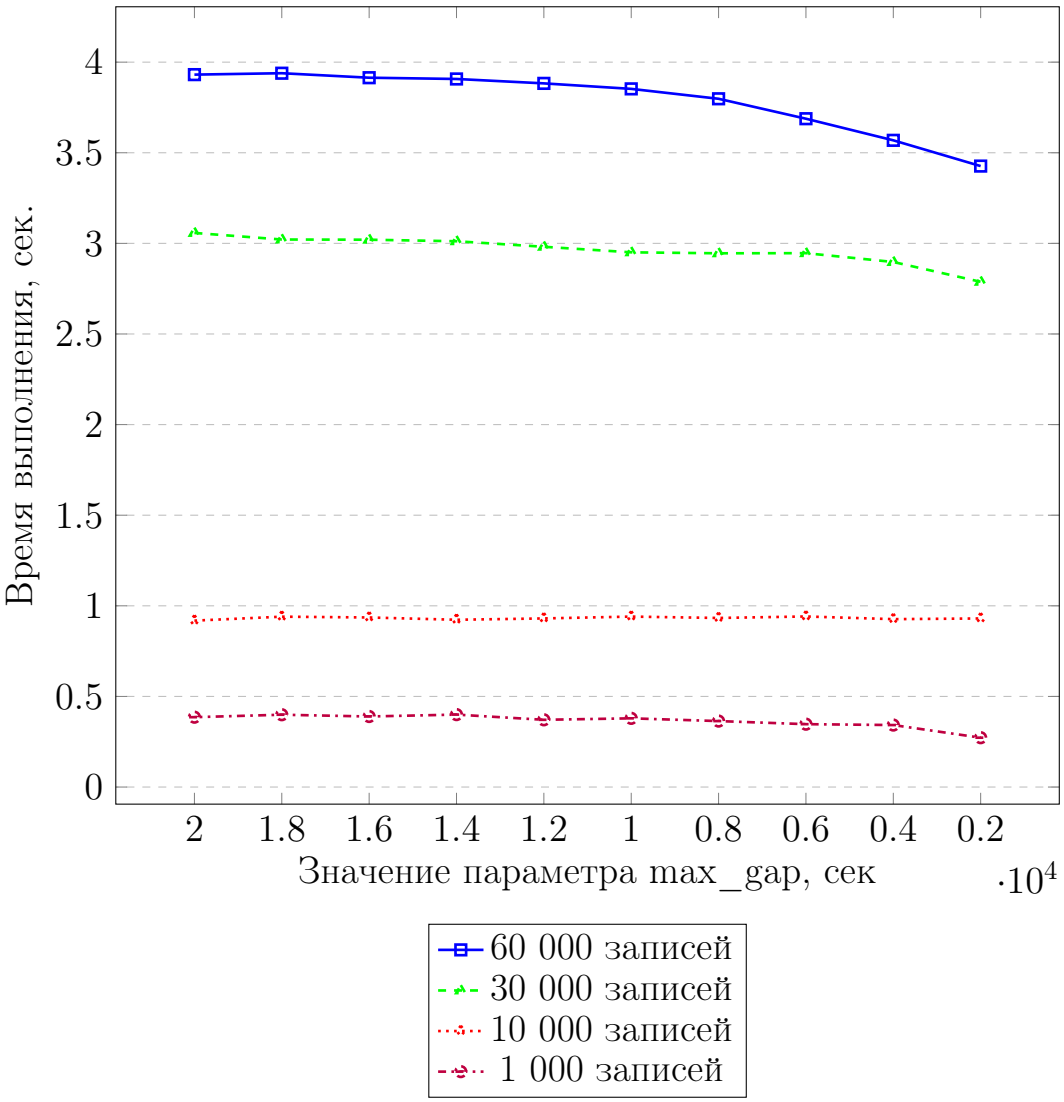

Рисунок 4.3 — Зависимость времени выполнения метода от максимального разрыва

4.2 Сравнительный анализ времени выполнения этапов метода

Чтобы провести сравнительный анализ времени выполнения этапов метода, замерялось их время выполнения с разными значениями минимальной поддержки и количеством записей 1000 раз, а затем делилось на количество замеров. Параметр min_gap был равен нулю, а max_gap имел максимально возможное значение На рисунке 4.4 представлен результат исследования в виде графиков.

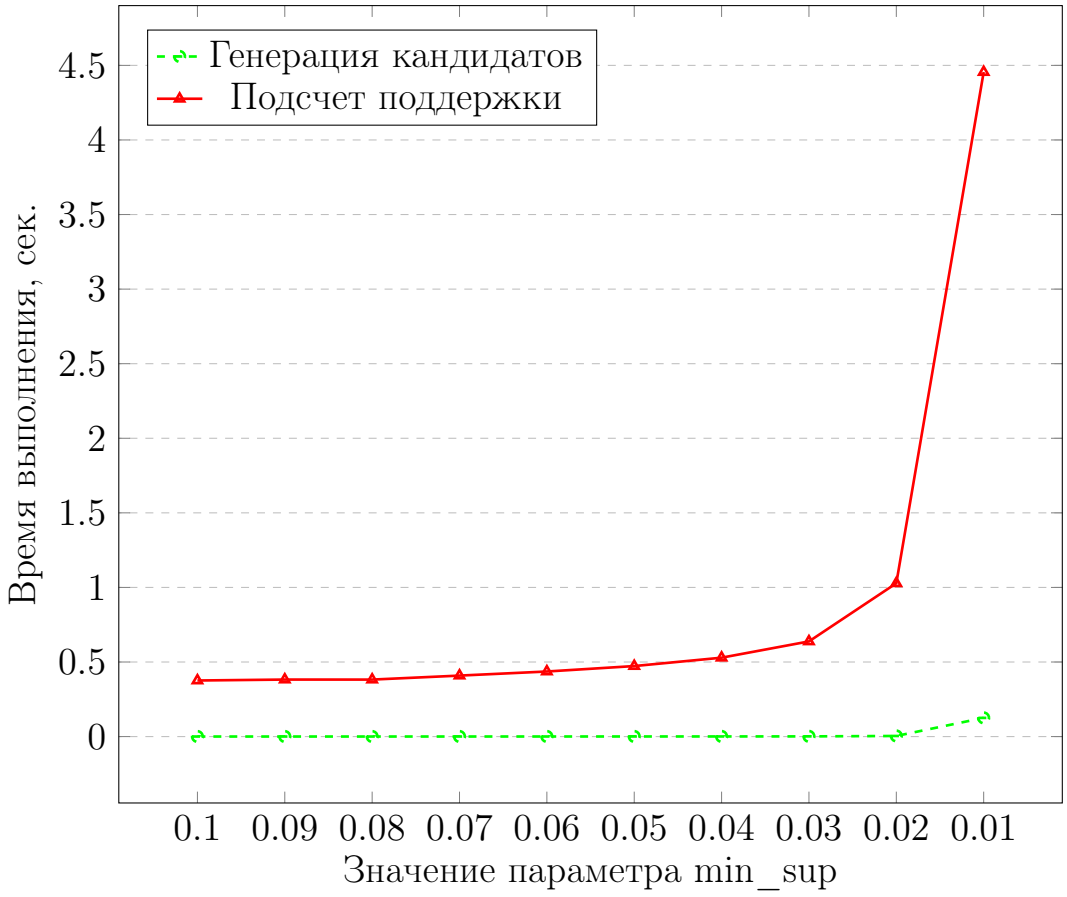

Рисунок 4.4 — Зависимость времени выполнения разных этапов метода от минимальной поддержки для 66788 записей

Вывод из исследовательского раздела

В данном разделе был проведен анализ времени выполнения метода в зависимости от параметров и времени выполнения его этапов.

Как и ожидалось, при уменьшении минимального уровня поддержки, время выполнения будет расти т.к. в таком случае больше последовательностей будут проходить отбор. Особенно видна разница между временем выполнения при значениях минимальной поддержки 0.02 и 0.01.

При увеличении минимального разрыва между командами, время выполнения уменьшается т.к. в таком случае получается меньше последовательностей из-за увеличения ограничения. К тому же, чем больше записей в базе данных, тем сильнее влияет изменения параметра min_gap.

При уменьшении максимального разрыва между командами, время выполнения тоже уменьшается, потому что в этой ситуации, также получается меньше последовательностей проходят ограничения.

В результате анализа времени выполнения разных этапов метода, можно сделать вывод, что подсчет поддержки кандидатов занимает большую часть времени, чем их генерация.

ЗАКЛЮЧЕНИЕ

По итогу проделанной работы была достигнута цель – разработан и программно реализован метод анализа активности пользователей САПР с использованием поиска последовательных шаблонов.

Также были решены все поставленные задачи, а именно:

— рассмотрены существующие решения в области анализа активности пользователей, выбраны для них критерии оценки и проведено сравнение;

— формализована задача в виде IDEF0-диаграммы;

— разработан метод анализа активности пользователей САПР с использованием поиска последовательных шаблонов;

— разработано программное обеспечение, реализующее описанный метод;

— исследованы характеристики разработанного метода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Siochi A.C., Ehrich R.W Computer Analysis of User Interfaces Based on Repetition in Transcripts of User Sessions. // ACM Transactions on Information Systems. – 1991. – Т. 9. – № 4. – С. 309–335.

2. Данилов Н.А., Шульга Т.Э. Метод построения тепловой карты на основе точечных данных об активности пользователя приложения // Прикладная информатика. – 2015. – Т. 10. – № 2. – С. 49–58.

3. Danilov N. Software usability evaluation based on the user pinpoint activity heat map. / Danilov N., Shulga T., Frolova N., Melnikova N., Vagarina N., Pchelintseva E. // Advances in Intelligent Systems and Computing. – 2016. – Т. 465. – С. 217–225.

4. Balbo S. Leading Usability Evaluations to WAUTER. / Balbo S., Goschnick S., Tong D., Paris C. // Proc. 11th Australian World Wide Web Conf. (AusWeb), Gold Coast, Australia, Southern Cross Univ. – 2005. – С. 279–290.

5. Swallow J., Hameluck D., Carey T. User interface instrumentation for usability analysis: a case study. // CASCON'97. – Toronto, Ontario. – 1997.

6. Shah I. Event patterns as indicators of usability problems. // Jour. of King Saud Univ., Comp. and Inform. Sci. – 2008. – С. 31–43.

7. Mabroukeh N.R., Ezeife C.I. A taxonomy of sequential pattern mining algorithms. // ACM Computing Surveys (CSUR). – 2010. – Т. 43. – № 1. – статья № 3.

8. Aloysius G., Binu D. An approach to products placement in supermarkets using prefixspan algorithm. // Jour. of King Saud Univ. Comp. and Inform. Sci. – 2013. – Т. 25. – № 1. – С. 77–87.

9. Сытник А.А. Математическая модель активности пользователей программного обеспечения. / Сытник А.А., Шульга Т.Э., Данилов Н.А., Гвоздюк И.В. // Программные продукты и системы. – 2018. – Т. 31. – № 1. – С. 79-84

10. Agrawal R., Imielinski T., Swami A.N. Mining Association Rules between Sets of Items in Large Databases. // Proceedings of the 1993 ACM SIGMOD international conference on Management of data. SIGMOD '93. – Washington, D.C., USA. – 1993. – T. 22(2). – C. 207-216.

11. Agrawal R., Srikant R. Fast algorithms for mining association rules // Proceedings of the 20th International Conference on Very Large Data Bases, VLDB. – Santiago, Chile. – 1994. – C. 487-499.

12. Zaki J Mohammed, Meira Jr Wagner. Data Mining and Analysis: Fundamental Concepts and Algorithms. – New York: Cambridge University Press, 2014. – C. 595

13. Agrawal R., Srikant R. Mining Sequential Patterns // Proc. of the 11th Int'l Conference on Data Engineering. – 1995. – C. 3–14.

14. Srikant R., Agrawal R. Mining Sequential Patterns: Generalizations and Performance Improvements // EDBT. Springer Berlin Heidelberg. – 1996. – C. 1–17.

15. Интеллектуальный анализ данных: учеб. пособие. // Томск: Издательский Дом Томского государственного университета, 2016. – С. 120

16. Card S., Moran T., Newell A. The keystroke-level model for user performance time with interactive systems. // Communications of the ACM. – 1980. – Т. 23. – № 7. – С. 396–410.

17. Бьёрн Страуструп Язык программирования C++ - специальное издание. Москва: Бином, 2010. 1136 с.

18. Qt documentation [Электронный ресурс] Режим доступа: `http://doc.qt.io/` (дата обращения: 01.12.2022)

ПРИЛОЖЕНИЕ А

```
#Logger version=1
#Product 'Платформа nanoCAD x64' version=22
2023-02-10 20:17:38 <Inspector>'Show'</Inspector:Completed>
2023-02-10 20:17:57 Document 'Без имени0' activated.
2023-02-10 20:18:06 <NewDocument>'Без имени0' to 'Без имени0'</NewDocument:Completed>
2023-02-10 20:18:06 <StartupVPerfTest></StartupVPerfTest:Completed>
2023-02-10 20:18:07 <TipOfDay></TipOfDay:Completed>
2023-02-10 20:23:34 <Line>Первая точка:15104.3,47663.8,0.0; Следующая
точка:44330.4,47663.8,0.0; Следующая точка:51683.4,29606.9,0.0; Следующая
точка:38931.9,17320.8,0.0; Следующая точка:13242.8,14435.4,0.0; Следующая
точка:2073.6,25511.6,0.0; Следующая точка:3469.7,25511.6,0.0</Line:Completed>
2023-02-10 20:23:41 Closing session.
2023-02-10 20:23:41 <Exit>msgBox:'Вы хотите сохранить изменения, произведенные в 'Без
имени0' ?':No; 'Без имени0'</Exit:Completed>
```

Рисунок 4.1 — Пример логов NanoCAD

ПРИЛОЖЕНИЕ Б

Листинг 4.1 — Класс LogReader

```
class LogReader
{
public:
  LogReader();

  static shared_ptr<LogReader> instance();

  void readLogs(QString dir_name = "./");
  void readLogsWithoutTime(QString dir_name = "./");
  void includeEndCmds(bool val = true);
  QStringList getIgnoreList() const;
  QStringList getNewSessionCmdsList() const;
  void setIgnoreList(QStringList list);
  void setNewSessionCmdsList(QStringList list);

private:
  void readFile(const QFileInfo& file_info, QList<QString> &commands, int
     &session_id);
  void readDir(const QString& abs_path, QList<QString> &commands, int
     &session_id);
  void readFileWithoutTime(const QFileInfo& file_info, QList<QString>
     &commands, int &session_id);
  void readDirWithoutTime(const QString& abs_path, QList<QString> &commands,
      int &session_id);
  int getTimeFromRecord(QString r);
  bool getCommandFromRecord(QString r, QString& res);
  QStringList getAllCommandsFromRecord(QString r);
  QStringList getTwoCommandsFromRecord(QString r);

private:
  QDir::Filters dir_filters;
  QStringList ignore_commands;
  QStringList new_session_commands;
  bool end_cmds;
};
```

Листинг 4.2 — Класс DataBase

```
enum Status
{
  OK = 0,
  EXEC_ERROR,
  EMPTY_RES,
  DATABASE_OPEN_ERROR,
  DATABASE_DOES_NOT_EXISTS,
  DATABASE_IS_NOT_VALID
};

class DataBase
{
public:
  DataBase();

  static shared_ptr<DataBase> instance();

  Status setSQLiteDataBase(QString db_name = "db_name");
  Status resetSQLiteDataBase();
  bool databaseExists(QString db_name);
  Status getRowsInLogs(QString db_name, int &rows_number);
  Status getSessionsInLogs(int &sessions_n);
  Status addCommand(int session_id, const QString& datetime, const QString
     &cmd, int &id);
  Status addCommand(int session_id, int int_time, const QString &cmd, int &id);
  Status getCmdsMap(QMap<int, QString>& cmds_map);
  Status getSessionsNum(int& sessions_num);
  Status getAllLogs(int commands_num, QList<Session>& sessions);

  QString lastError();

private:
  inline Status execQuery(QString query);

private:
  QString m_last_error;
  QString cur_db_name;
  QSqlQuery m_query;
};
```

Листинг 4.3 — Класс Calculator

```
class Calculator
{
public:
  Calculator();

  QList<Sequence> getFrequentSequences(double _min_sup = -1, int _min_gap = -1,
      int _max_gap = -1);
  void printFrequentSequences();
  QString getSeqStr(const Sequence &seq);
  void setSameCmds(bool val);

private:
  void prepareGSP();
  QList<Sequence> generateCandidates1();
  QList<Sequence> generateCandidates();
  bool findCommand(int cmd, const Session& session, int min_time, int
      prev_cmd_id, int &time, int &id) const;
  bool sessionSupportsSequence(const Session& session, const Sequence& seq);
  Sequence findFreqSequenceByCommand(int cmd);
  double calcLift(Sequence seq);
  int countSupport(QList<Sequence> &candidates, const QList<Session> &sessions);
  void sortFrequentSequences();

private:
  QList<Sequence> freq_seqs;
  QList<Sequence> cur_freq_seqs;
  QMap<int, QString> cmds_map;
  int sessions_count = 0;
  QString db_file_path;

  double min_support = 0.5;
  int min_gap = 0;
  int max_gap = INT_MAX;
  bool same_cmds = true;
};
```

ПРИЛОЖЕНИЕ В

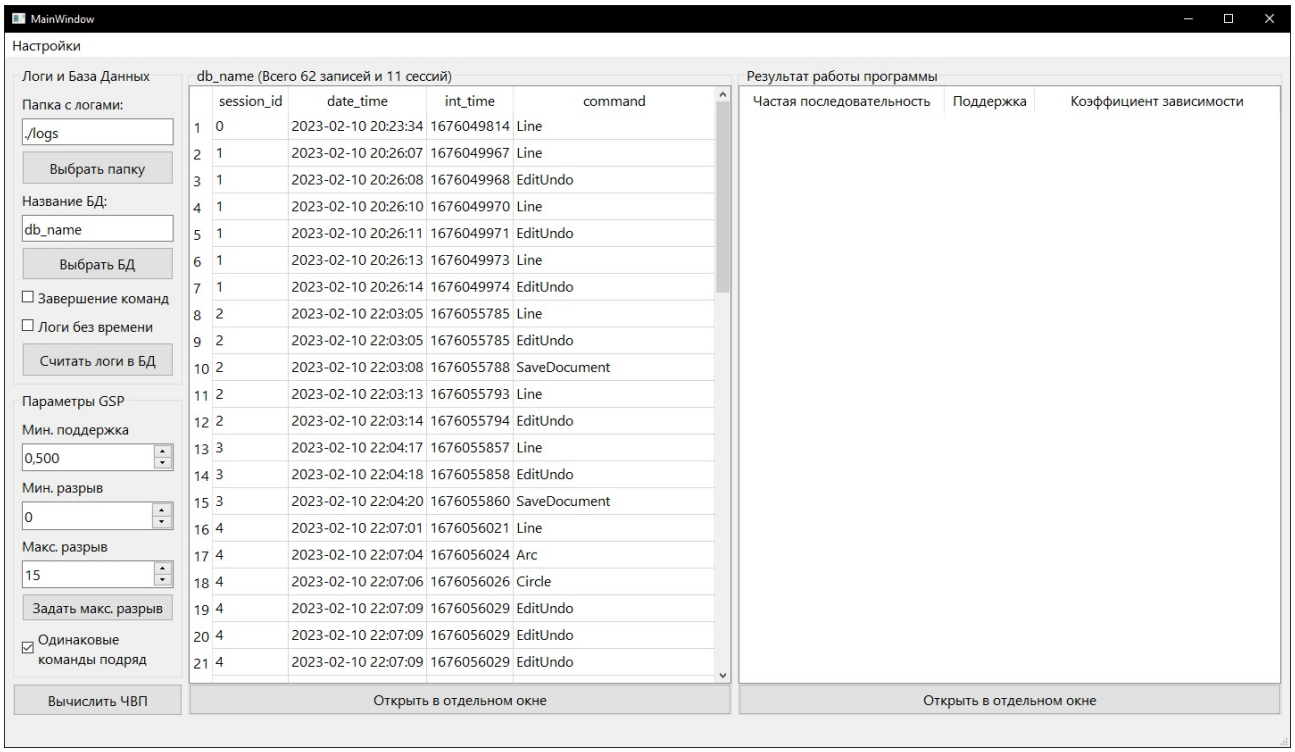

Рисунок 4.1 — Интерфейс программы 1

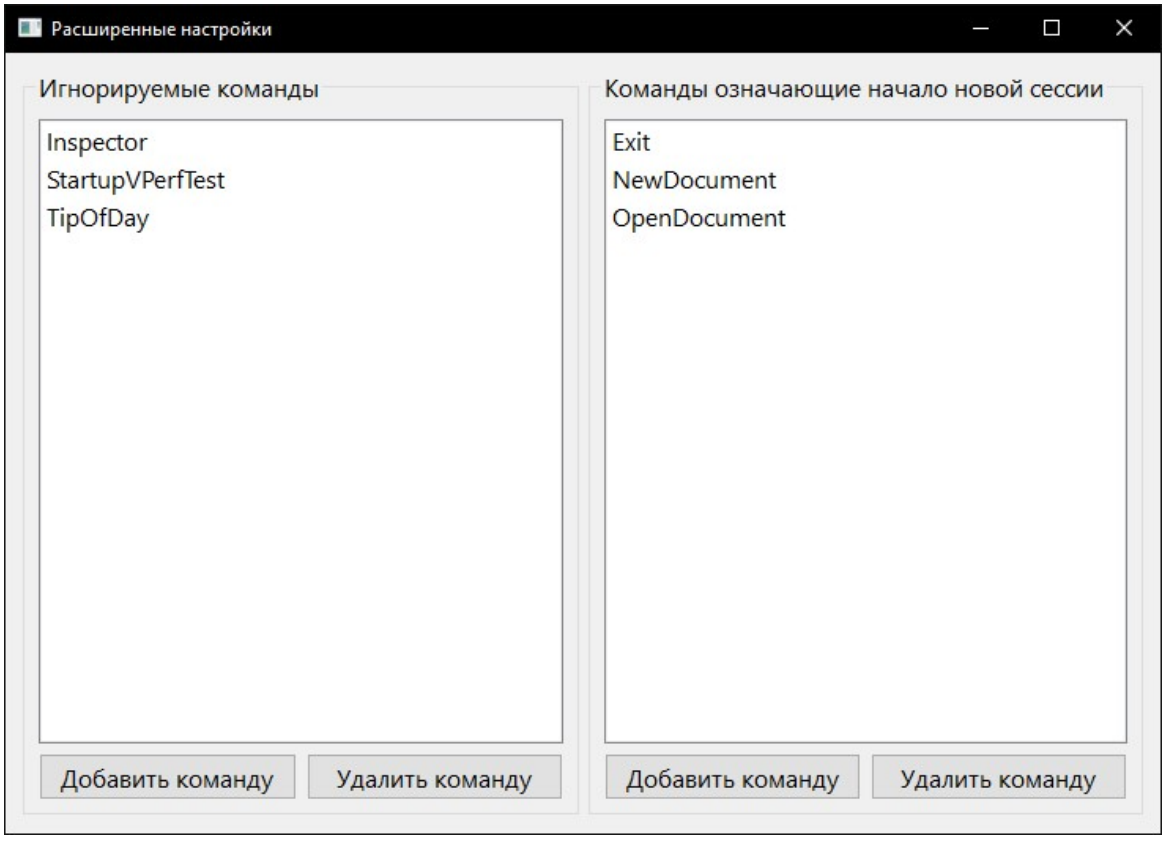

Рисунок 4.2 — Интерфейс программы 2

ПРИЛОЖЕНИЕ Г

Презентация