Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра информатики

Струсь Глеб Игоревич

Постановка и решение изобретательских задач в программировании на основе методов ТРИЗ

Дипломная работа

Допущена к защите.

Зав.кафедрой:
д.ф.-м.н., проф. Косовский Н.К.

Научный руководитель:
ст.п. Одинцов И.О.
ст.н.с. Рубин М.С.

Рецензент:
д.ф.-м.н., проф. Косовский Н.К.

Санкт-Петербург

2010

Saint-Petersburg State University

Faculty of Mathematics & Mechanics

Computer science department

Strus Gleb Igorevich

Definition and solving inventive problems in programming using TRIZ methods

Graduate work

Allowed to defence.

Head of a chair:
Dr. of Phys. and Math., Prof. N. K. Kosovskii

Scientific adviser:
Senior lecturer I. O. Odintsov

Senior staff scientist M. S. Rubin

Reviewer:
Dr. of Phys. and Math., Prof. N. K. Kosovskii

Saint-Petersburg

2010

Оглавление

Введение

Со времени своего появления, программирование постоянно развивается. Этот процесс связан не только с появлением новых языков программирования, технологий или аппаратных средств, но и с комплексом задач, решаемых с его помощью. Именно появление новых задач, для решения которых необходимо прибегать к алгоритмизации и автоматизации, диктует постоянный вектор развития. Первые задачи, решаемые с помощью программирования, были просты и требовали линейных алгоритмов. Автоматизация же сводилась к простому выполнению одного из них. В настоящее время сложность задач многократно возросла. Написание некоторых программных продуктов требует усилий многих квалифицированных специалистов. Их же выполнение - огромных вычислительных мощностей. Также повысились требования к надежности и функциональности программных продуктов.

Вместе с увеличением сложности задач, увеличились и риски. Большая их часть связана с возникающими в процессе программирования противоречиями. Создание самого алгоритма решения задачи часто представляет определенную сложность. Кроме того, в процессе создания программного продукта возникают и другие, не менее важные, проблемы и конфликтные ситуации.

ТРИЗ, или теория решения изобретательских задач, является одной из ведущих методик совершенствования изобретений, ускорения их создания и решения технических противоречий. Изначально, ТРИЗ была ориентирована на решение изобретательских задач в технической области. В результате своего развития ТРИЗ вышла за ее пределы и сейчас используется и в нетехнических областях. Информационные технологии и программирование в частности являются одними из важнейших и наиболее быстро развивающихся отраслей современной науки. Поэтому использование ТРИЗ

для ускорения создания алгоритмов и разрешения противоречий в программировании представляет наибольший интерес.

На сегодняшний день сама ТРИЗ и методики, основанные на ней, приобретают все большую популярность в мире. Это обусловлено, в первую очередь, их высокой эффективностью и надежностью. География ТРИЗ охватывает основные развитые страны: США, ЕС, Японии, Южной Корее, России, Китай. Российская компания ЦИТК “Алгоритм”, работающая совместно с американской фирмой GEN3 Partners, является одним из лидеров в предоставлении инновационных решений на мировом рынке.

Целью данной работы является обоснование применимости методов ТРИЗ для решение задач в программировании и, в случае необходимости, адаптация под специфику предметной области. Также, ставится задача создания программного продукта, который бы автоматизировал определенные алгоритмы этой методики.

Объектом работы является методика ТРИЗ и практические методы постановки и решения изобретательских задач, разработанные в GEN3 и ЦИТК "Алгоритм".

Предметом работы является алгоритм решения изобретательских задач (АРИЗ) и алгоритмы постановки и анализа изобретательских задач.

Следует выделить основные задачи данной работы:

1. Исследовать Алгоритм Решения Изобретательских Задач (АРИЗ)
2. Обосновать применимость АРИЗ для решения изобретательских задач в программировании
3. Исследовать алгоритмы постановки изобретательских задач
4. Определить методы анализа систем, необходимые для осуществления алгоритмов постановки изобретательских задач

5. Адаптировать выбранные алгоритмы и методы для программной реализации
6. Создать программный продукт, основанный на выбранных алгоритмах

Обоснование применимости методов ТРИЗ для решения изобретательских задач в программировании позволит начать их широкое использование в профессиональной сфере. Программный продукт, работа которого будет основана на алгоритмах этой методики, позволит в значительной степени упростить и ускорить работу не только программистов при создании программных продуктов, но и специалистов по ТРИЗ при решении ими изоретательских задач.

Глава 1. Постановка задачи и ее актуальность

1.1. Постановка задачи

Следует выделить две наиболее важные задачи данной работы.

1. Основной целью данной работы является аналитическое исследование применения АРИЗ для решения изобретательских задач в программировании. Оно сочетает в себе элементы как из технических, так и из менее строгих и формальных областей. С одной стороны, можно рассматривать выполнение программ и алгоритмов на аппаратном уровне. Тогда основными элементами будут технические компоненты, например, монитор или процессор. С другой стороны, программирование содержит также и нематериальную составляющую. У всякого программного продукта есть логика выполнения. На этом уровне программист имеет дело с объектами, логическими и другими формальными правилами или переменными. Такая двойственная природа программирования дает

основания полагать, что для решения возникающих в нем противоречий подойдут методы, работающие в технических областях. Однако, эта же двойственность не позволяет просто использовать их без проведения необходимой проверки их пригодности.

Таким образом, необходимо на примере конкретных задач из области программирования показать эффективность данного алгоритма, его универсальность. Для этого будут использованы различные задачи из области программирования, решение которых уже имеется. Это позволит проверить, соответствует ли решение, полученное с помощью АРИЗ, практическому. Кроме того, будут рассмотрены оба класса изобретательских задач – те, которые касаются аппаратного уровня, и те, которые касаются логического.

Так как программирование представляет собой обширную область знаний, то сложность задач может сильно варьироваться. Их форма может быть как вполне конкретной, так и довольно общей. В последнем случае возникает ряд проблем, связанных со слишком широким кругом знаний, требуемых для их решения. Кроме того, часто бывает трудно формализовать условие задачи, сформулировать искомое или даже выделить в исходной задаче какие бы то ни было отдельные части. Поэтому начальный этап постановки и формулировки изобретательской задачи является в данном случае особенно важным. Алгоритм будет применяться в несвойственной для себя области, поэтому корректность его работы будет сильно зависить от правильности и точности входных данных. В связи с этим ставится задача анализа и систематизации вспомогательных алгоритмов, направленных на получение максимально полных и верных данных о задаче. Эти алгоритмы должны быть автоматизированы. Вследствие сложности процесса постановки задачи, большого разнообразия алгоритмов, а также того, что программистам данные алгоритмы вероятно будут незнакомы, их программная реализация

позволит не только упростить применение, но и сведет риск появления ошибок на данном этапе решения задачи к минимуму.

2. Поэтому еще одной важной задачей данной работы является создание программного продукта, который бы реализовывал некоторые алгоритмы постановки и анализа изобретательских задач.

1.2. Причины возникновения проблемы и ее актуальность

Начальный вариант ТРИЗ был основан Генрихом Сауловичем Альтшуллером и его коллегами в 1946 году, и впервые опубликован лишь в 1956 году. Область технических знаний на тот момент имела наибольшее значение в экономике, в частности, и в мире вцелом. Тогда появление ТРИЗ было вызвано потребностью ускорить изобретательский процесс, исключив из него элементы случайности: внезапное и непредсказуемое озарение, слепой перебор и отбрасывание вариантов, зависимость от настроения и т. п.. Это позволило сделать его более прогнозируемым и контролируемым, повысить качетсво изобретений. ТРИЗ хорошо зарекомендовал себя. В результате своего развития он вышел за пределы решения изоретательских задач в технической области и стал применяться в некоторых нетехнических областях, таких как бизнес, педагогика или политика.

В современном обществе все больший вес набирают информационные технологии. Последний этап научно-технической революции привел к осознанию важности информации и ее влияния на все аспекты жизни общества. Поэтому программирование представляет собой наиболее интересную нетехническую область для применения развившейся методики ТРИЗ. Характерно, что его темпы развития сейчас сравнимы с темпами развития техники в конце 20 века. Такому стремительному развитию непременно сопутствует возникновение и решение сложных задач. И чем

эффективнее данные задачи будут решаться, тем больше будет увеличиваться скорость прогресса.

Сегодня почти ни один проект по разработке программного обеспечения не может быть закончен вовремя. Одной из главных причин этого служат трудности, возникающие в процессе создания программного продукта. Они могут возникнуть на любом этапе. Неправильное решение при проектировании архитектуры скажется в процессе реализации и повергнет весь проект в хаос. Найденная ошибка на этапе тестирования задержит выход системы на рынок, так как потребует слишком много времени на устранение. При этом они представляют высокую опасность именно в силу своей непредсказуемости, заставляя ошибаться раз за разом на этапе планирования.

Одной из основных задач ТРИЗ является устранение элемента случайности в изобретательском процессе. Также она служит цели ускорения этого процесса. Оба этих качества ТРИЗ могут быть очень полезны при создании программного продукта. Поэтому обоснование возможности применения ее методов в программировании представляет высокий интерес и может принести практическую пользу.

Большая часть методов ТРИЗ направлена на решение изобретательских задач и устранение противоречий. В технической сфере теория доказала свою состоятельность. Ее методы разрешения технических противоречий являются высоко эффективными и надежными. Вместе с тем, противоречия в программировании столь же обычное и даже более частое явление. Многие задачи содержат в себе скрытые или явные технические противоречия. Например, можно ускорить работу алгоритма, но тогда для его работы потребуется больше памяти, можно реализовать всю функциональность, но программа станет слишком большой и сложной, ее будет неудобно сопровождать.

По своей сути, технические противоречия в программировании не отличаются от таковых в технической области. При ускорении работы алгоритма мы улучшаем параметр “время выполнения”, однако это приводит к ухудшению параметра “требуемый объем памяти”. Такое сходство дает основания полагать, что если изобретательские задачи похожи, то и методы, пригодные для решения одних, подойдут и для решения других.

Изобретательские задачи являются неотъемлемой частью программирования. Более того, можно сказать, что их наличие диктуется самой его природой. Написание программ для первых вычислительных машин было очень трудной задачей. Тогда ограничения накладывались на программиста абсолютно всем, что он использовал в своей работе. Первые языки программирования были слишком далеки от естественных языков и слишком близки к машинным. Это приводило к огромному объему кода, необходимого для реализации даже простых функций. В то же время, несовершенное аппаратное обеспечение накладывало ограничения скорости выполнения команд, устройства хранения данных не позволяли делать программы слишком большими. В то время процесс написания программы представлял собой можество противоречий, которые приходилось решать, используя ограниченные ресурсы.

Если разрешение противоречий было настолько рутинной деятельностью для программистов в прошлом, то возникает вопрос, почему когда они сталкиваются с ними сейчас, на решение задачи уходит много времени или задача оказывается не решаемой. Одним из возможных объяснений этого мог бы служить факт изменения характера или природы противоречий. Поскольку проверить это не представляется возможным, то следует искать объяснение не в самом программировании. Существуют объективные причины, связанные не с противоречиями, а с мышлением программистов.

Перед современным программистом сейчас возникает меньше противоречий, чем перед его коллегой двадцать лет назад. Это связано с тем, что мы привыкли называть шаблонами образцовые решения различных задач. Паттерны в объектно-ориентированном программировании, быстрые алгоритмы сортировки массивов, поиска элементов - всем этим программисты пользуются сейчас постоянно. При этом противоречие, лежащее в основе породившей это решение задачи, скрыто от нашего восприятия.

Широкое распространение баз знаний также сыграло немаловажную роль. Во многих компаниях существуют специальные библиотеки фрагментов кода, реализующих типовую функциональность. Когда возникает необходимость решить какую бы то ни было задачу, программист всегда может обратиться к своим коллегам или поискать решение в интернете. Вполне вероятно, что ему удастся найти решение, так как кто-нибудь уже сталкивался с похожей проблемой.

К сожалению, насколько бы простым не был такой способ, он влечет за собой множество недостатков. При такой системе, в отсутствие хороших решений, происходит не поиск хороших, а тиражирование плохих существующих. Также это переключает настройку мышления программиста с решения текущей задачи на поиск аналогичной. Это не является недостатком само по себе, однако, в свете первого факта становится не самой лучшей практикой.

Если же решение найти не удается, то программист часто придумывает далеко не самый лучший способ, тем самым добавляя его к существующим плохим решениям.

Никто не учит думать программиста в терминах технических противоречий. Он может чувствовать противоречие, но не может сформулировать или, хотя бы, локализовать его. Методы ТРИЗ могут стать

надежным инструментом при решении изобретательских задач в программировании.

То, что с изобретательскими задачами программисты сталкиваются сейчас реже, не снизило важность их разрешения. Каждая такая задача теперь требует особого внимания, так как часто от нее зависит успешность всего проекта. Почти любая серьезная проблема, возникающая в процессе создания программного продукта, может рассматриваться в качестве изобретательской задачи. В рамках ТРИЗ существует набор инструментов, одним из которых является АРИЗ, позволяющих решать такие задачи. Поэтому если будет продемонстрировано, что методы ТРИЗ применимы в програмировании, то это может стать серьезным шагом в направлении улучшения качества программных продуктов и ускорения прцесса их разработки.

1.3 Краткий обзор существующих подходов к проблеме

На данный момент направление применения методов ТРИЗ для решения изобретательских задач в программировании является новым. Хотя многие компании предлагают консультационные услуги по внедрению инновационных технологий, они отстраняются от реального решения задач в области информационных технологий. В основном, такими компаниями предоставляются справочные материалы и проводятся тренинги персонала.

За рубежом ТРИЗ получила довольно широкое распространение, однако там она находит больше практическое применение и утрачивает теоретическую составляющую. Поэтому, несмотря на наличие интереса к ней, отсутствуют научные подходы к рассматриваемой проблеме. Интереснее всего с точки зрения развития ТРИЗ в программировании выглядит работа Джона Стэйми “ТРИЗ и Экстремальное программирование”[1]. В ней автор проводит сравнительный анализ методологии экстремального программирования и ТРИЗ, определяет сходства в их методах и инструментах, проводит аналогии между 12 основными принципами

экстремального программирования и изоретательскими принципами из ТРИЗ. Дальнейший вектор развития автор видит в более подробном анализе процессов экстремального программирования с точки зрения ТРИЗ.

Наибольшее же распространение попытки использования ТРИЗ в программировании получили в России. При этом они делятся на два типа.

Первые - это статьи и материалы единичного характера. Они отличаются невысокой строгостью, неформальностью и слабой систематизируемостью. Сюда можно отнести различные темы форумов сообщества программистов, например ветвь форума русского сообщества разработчиков программного обеспечения посвященного философии программирования[2]. Из анализа этих материалов можно сделать вывод, что интерес к данному вопросу среди профессионалов достаточно высок. Осуществляются попытки применения методов ТРИЗ в программировании, однако основной проблемой является слабая подготовленность программистов в области ТРИЗ.

Вторые - это систематическое изучение данного вопроса, выраженного в цикле статей. Следует отметить работы С.Сычева и К. Лебедева[3]. В них проводится анализ конфликтных ситуаций в программировании и показываются варианты решения, затем объясняемых в базовых терминах ТРИЗ. Главными недостатками этих статей являются их иллюстративный характер и применение отдельных инструментов ТРИЗ.

Последние работы в этой области связаны с возросшим интересом компании Intel к применению методов ТРИЗ для повышения эффективности разработки программных продуктов. В статье И.О. Одинцова и М.С. Рубина[4], опубликованной на конференции Международной Ассоциации ТРИЗ в 2009 г., рассматривается возможность применения методов теории для разработки программного обеспечения и даются некоторые примеры попыток практического их применения. Данная статья служит не только

некоторой отправной точкой, но и обозначает возрастающий интерес к применению ТРИЗ в программировании.

Программное обеспечение, основанное на методах ТРИЗ, в России распространено меньше, чем за рубежом. Главным поставщиком таких программных продуктов на нашем рынке является компания “Метод”[5]. Иностранные компания, предлагающие похожее ПО, часто выводят на рынок инструменты, в основе которых лежит консультационный механизм. Одним из наиболее известных программных продуктов является TechOptimizer. Этот продукт фирмы Invention Machine прошел ряд улучшений и сейчас предлагается под названием Goldfire Innovator[6]. Он интересен, в первую очередь, широким кругом методов ТРИЗ, которые поддерживает. Однако, каждый из этих методов реализован больше в консультационной форме, нежели действительно автоматизирован. Goldfire предлагает инструменты для решения задач, но не помогает ими пользоваться. Данный программный продукт больше ориентирован на поддержку процесса разработки инноваций путем предоставления обширных баз знаний и большого количества справочного материала. Таким образом, существующие на данный момент программные продукты, основанные на методах ТРИЗ, не могут рассматриваться в качестве автоматизированных. Они лишь являются вспомогательными средствами при работе по алгоритмам ТРИЗ.

1.4 Основные описания

Приведенные формальные определения взяты из глоссария ЦИТК “Алгоритм”[7] и дополнены в соответствии с теоретической составляющей работы.

Алгоритм решения изобретательских задач (АРИЗ) - Инструмент для решения задач, который трансформирует сложную ситуацию в четко сформулированную модель задачи, которая может быть успешно решена с помощью инструментов ТРИЗ.

Элемент-Полевая Модель (Элепольная модель, Элеполь) - Символическая модель задачи или ее решения, сформулированная в виде взаимодействий между Элементами и Полями (виртуальными, реальными или усовершенствованными).

Идеальный Конечный Результат - Это ситуация, когда нужное действие получается без каких-либо затрат (потерь), усложнений и нежелательных эффектов.

Изобретательские Принципы (Приемы) - Обобщенные рекомендации по изменению системы с целью решения изобретательской задачи, сформулированной в виде противоречия (Изобретательский Принцип - это абстрактная модель решения задачи).

Матрица Альтшуллера - Инструмент для решения изобретательских задач, рекомендующий Изобретательские Приемы для разрешения Противоречий.

Оперативная зона - Область пространства, в котором выполняется одно из действий Противоречия или должно быть выполнено одно из требований Физического Противоречия.

Оперативное время - Промежуток времени, в ходе которого выполняется одно из действий Противоречия или должно быть выполнено одно из требований Физического Противоречия.

Противоречие - Ситуация, при которой попытка улучшить один Параметр Системы ведет к недопустимому ухудшению другого Параметра.

Типовые Параметры - Ограниченный набор обобщенных Параметров, в улучшении которых, как правило, нуждаются Системы и которые перечисленны в Матрице Альтшуллера.

Глава 2. Обоснование применимости методов ТРИЗ для задач в программировании

2.1. Применение АРИЗ для решения изобретательских задач в программировании

Одним из главных инструментов ТРИЗ для решения изобретательских задач является АРИЗ. Алгоритм решения изобретательских задач (АРИЗ) - комплексная программа алгоритмического типа, основанная на законах развития технических систем и предназначенная для анализа и решения изобретательских задач. АРИЗ возник и развивался вместе с Теорией Решения Изобретательских Задач (ТРИЗ) [8]. Одной из главных особенностей алгоритма является параметрический подход для анализа и решения изобретательских задач[9]. Он заключается в выделении типовых параметров системы и разрешении противоречия путем использования изобретательских принципов, выбор которых может быть определен по Матрице Альтшуллера.

Впервые словосочетание "алгоритм решения изобретательских задач" использовано в приложении "Технико-экономические знания" к еженедельнику "Экономическая газета" за 1 сентября 1965 г. Аббревиатура АРИЗ впервые использована в книге Г.С. Альтшуллера "Алгоритм изобретения", Московский рабочий, 1-е изд.: 1969, 2-е изд.: 1973. В дальнейшем модификации АРИЗ включали указание на год публикации, например АРИЗ-68, АРИЗ-71.

Автором АРИЗ является Г.С. Альтшуллер. Однако, при разработке последних модификаций алгоритма (АРИЗ-77, АРИЗ-82, АРИЗ-85) были учтены замечания и рекомендации многих специалистов по ТРИЗ. В своем выступлении в 1986 году Г.С. Альтшуллер задал вектор будущего развития АРИЗ, отметив, что «Анализ развития алгоритма поможет конструировать

алгоритмические программы решения неизобретательских (научных, художественных) задач и проч.» [10].

Современный АРИЗ используется для решения сложных задач. Он используется тогда, когда решение не может быть найдено другим способом, или же решение существует, но является невыполнимым. Часто, такие задачи не имеют четкой формулировки, их исходные данные могут быть неполными или искаженными. Более того, может быть даже неверно указан требуемый результат. Все эти черты современных технических задач в полной мере присущи и изобретательским задачам из области программирования.

На сегодняшний день наиболее широко используется АРИЗ-85-В. Он является последней модификацией, разработанной самим Альтшуллером. За более чем двадцатилетнюю историю своего существования, предпринималось множество попыток развития АРИЗ-85-В. Неоднократно выделялись его основные недостатки [11]:

1. Отсутствует часть АРИЗ, точно определяющая исходную (изобретательскую) ситуацию.
2. Отсутствует часть АРИЗ, определяющая все возможные направления решения задачи и выбор наилучшего.
3. Способы разрешения физического противоречия (ФП) недостаточны и не детализированы.
4. Отсутствует система определения способа разрешения ФП в зависимости от его вида.
5. Включение части 4 в АРИЗ-85-В привело к разрыву логики АРИЗ.
6. Части 6-8 АРИЗ-85-В не используются при решении задач. Их необходимо развить и структурировать.

Однако, до настоящего момента не было ни одной версии алгоритма, которая бы смогла заменить версию 85 года. Проблема состоит в том, что

почти все предлагаемые кандидаты на роль нового алгоритма решения изоретательских задач были получены путем углубления в техническую область. Хотя, первоначально, ТРИЗ была создана на сугубо практической основе и базировалась на анализе большого числа технических патентов, со временем она стала эволюционировать и преобретать все более сильную теоретическую составляющую. АРИЗ, как метод этой теории, должен повторять ее путь. То есть будущее этого алгоритма должно заключаться в его универсализируемости, расширении области применения, что понимал Г.С. Альтшуллер.

На последнем ежегодном саммите разработчиков ТРИЗ была представлена статья В.М. Петрова и М.С. Рубина[12], в которой в очередной раз поднялась проблема создания АРИЗ нового поколения. Подход, описываемый в ней, не только предусматривает устранение существующих недостатков, но также предполагает расширение применения АРИЗ на нетехнические системы. Также отмечается необходимость создания части алгоритма, отвечающей за выбор и постановку исходной задачи. Последнее особенно актуально, потому что мы видели, что изобретательские задачи в программировании могут отличаться по форме от классических изобретательских задач и содержать скрытое противоречие.

Задачей данной работы является обоснование применения АРИЗ для решения изоретательских задач в программировании. Для этого будет решено несколько задач различного уровня сложности. Выход с помощью АРИЗ на решение будет означать возможность его применения и корректность для решения других задач. Будет использоваться последняя версия АРИЗ - АРИЗ 2010 (Алгоритм совершенствования систем 2010)[13,14]. Данный вариант является последней разработкой в направлении расширения применимости алгоритма на нетехнические системы и, вследствие этого, соответствует целям данной работы.

1. Задача об ускорении сортировки

Данная задача рассматривается как базовая. Ее можно сформулировать так - дан массив целых чисел, который необходимо отсортировать. Существует определенный алгоритм сортировки - сортировка методом пузырька. Время работы данного алгоритма квадратично зависит от длины массива. То есть для длинных массивов время работы алгоритма слишком велико. Необходимо, не создавая нового алгоритма сортировки, ускорить работу существующего. При этом требование использовать текущий алгоритм связано с тем, что создание нового алгоритма всегда представляет трудности и, по сути, является также изобретательской задачей.

Полное решение задачи по АРИЗ 2010 приведено в приложении 2.1.

АРИЗ выводит на контрольное решение - разбить исходный массив на части. После сортировки каждой отдельной части их следует соеденить обратно в цельный массив. При этом время на соединение упорядоченных массивов линейно зависит от их длины.

Интерес представляет предельный случай, когда массив разбивается на максимально мелкие части. Тогда сортировка каждой такой части, представляющей массив из двух элементов, сводится к перестановке местами двух элементов (если это требуется). Похожий принцип дробления лежит в основе алгоритма быстрой сортировки (quick sort), время работы которого логарифмически зависит от длины массива.

Получившееся по АРИЗ решение действительно позволяет сократить время работы алгорита сортировки пузырьком при большой длине массива. Оно является работоспособным и при анализе дает возможность выйти на получение нового алгоритма сортировки.

2. Задача о режимах редактирования фигур в геометрическом редакторе.

Данная задача рассматривается в качестве более сложной, поскольку затрагивает не только информационный аспект, но и физический (курсор мыши и отображение фигуры на экране монитора). Формулировка - необходимо реализовать работу двух различных режимов редактирования фигуры, обеспечив удобство их использования.

Полное решение задачи по АРИЗ 2010 приведено в приложении 2.2.

Решение, на которое выводит АРИЗ, является контрольным. Изменение режимов в зависимости от местаположения курсора мыши позволяет сделать их выбор легким и естественным для пользователя. Также это снижает когнитивную нагрузку, так как, по сути, такой подход вместо двухрежимной системы создает монорежимную. Данный принцип используется в некоторых современных редакторах трехмерной графики.

3. Задача о защите общедоступной программы.

Данная задача содержит наиболее острое противоречие, разрешение которого без специальных инструментов представляет наибольшую сложность. Также, она имеет высокую степень актуальности. Похожие задачи часто встречаются в современной практике программирования.

Формулировка задачи - необходимо защитить общедоступную программу от несанкционированного доступа. Ограничение в общедоступности накладывается внешней средой, например правилами организации. При этом должна быть сохранена конфиденциальность использования, то есть работать с программой должен только автор.

Полное решение по АРИЗ 2010 приведено в приложении 2.3.

В данной случчае АРИЗ вновь приводит к контрольному решению. Одним из ресурсов системы являются входные данные. Формат - их параметр, которые следует изменить. Создание специфического формата входных данных, известного только автору, позволяет надежно защитить программу, несмотря на то, что сама по себе она общедоступна.

В процессе работы алгоритма было получено еще одно решение. Можно отделить от программы некоторую часть, без которой она не сможет выполнять свою главную функциональность. В рассматриваемой задаче оно было признано неудовлетворительным в силу определенных недостатков. В частности автору программы было бы необходимо всегда иметь при себе требуюмую часть программы. Тем не менее, следует подчеркнуть важность наличия двух решений.

Некоторые наиболее сильные методы современной защиты работают по схеме “что-то, что ты имеешь, что-то, что ты знаешь” (something you have, something you know). Он основан на том, что для доступа к чему-либо нужно подтвердить свои права двумя способами - чем-то, что известно только санкционированному пользователю (например пароль), и чем-то, что у него есть физически (отпечаток пальца, USB флэш-накопитель). Контрольное решение данной задачи по своей сути представляет вторую часть приведенного принципа. Решение же, отброшенное в процессе, есть иллюстрация первого принципа. Таким образом, как и в первой задаче, здесь АРИЗ не только выводит на контрольное решение задачи, но и обозначает пути создания новой системы.

Необходимо отметить, что при решении приведенных задач была использована сокращенная таблица применения приемов устранения противоречий, предоставленная М.С. Рубиным. Из нее, по сравнению с Матрицей Альтшеллера, исключены параметры, которые относятся только к техническим системам. Во время решения сокращенная таблица позволяла сузить направление поиска и отсечь некоторые неверные варианты. Вместе с

тем, в сложных задачах сокращенная таблица может несколько затруднить поиск решения, так как в рассмотрение не попадут некоторые технические параметры. Однако, данное замечание справедливо только в том случае, если решение проводит квалифицированный специалист в области ТРИЗ. Для обучения и решения задач неспециалистами в этой области применение сокращенной таблицы носит преимущественный характер.

На основе этих трех задач, различающих по своей сложности и области возникновения, было показано, что АРИЗ 2010 может быть использован в качестве инструмента для решения изобретательских задач.

2.2. Алгоритмы постановки и анализа изобретательских задач

В рассмотренных в предыдущем разделе задачах формулировка проблемы и противоречия не представляли особых трудностей и могли быть сделаны без применения специальных методов.

Однако в программировании встречаются ситуации, когда постановка изобретательской задачи может не быть очевидной. Для этого в рамках ТРИЗ присутствует набор методов и алгоритмов, направленных на анализ ситуации, локализацию противоречия, выбор и постановку изобретательской задачи.

Важной задачей данной работы является создание системы, реализующей некоторые из этих алгоритмов. Поэтому был проведен комплексный обзор соответствующих методов ТРИЗ. В результате были выбраны алгоритмы, которые легли в основу созданного программного продукта.

За основу был взят алгоритм проведения свертывания системы, а также необходимые для его использования алгоритмы – компонентно-структурный анализ, функциональный анализ, причинно-следственный анализ[15]. При

применении к некоторой системе алгоритм свертывания позволяет получить ее функционально-идеальную модель, то есть комплекс необходимых функций системы, реализуемый минимальным количеством элементов этой системы.

Специалистом ЦИТК “Алгоритм” Герасимовым О.М. был предоставлен пример реального использования указанных методов – анализ и свертывание системы датчика температуры.

Наряду с решением изобретательских задач в программировании с помощью АРИЗ, было важно показать применимость методов постановки и выбора задач в области программирования.

Задача о программе вычисления произвольного полинома

Имеется программа, которая позволяет найти значение произвольного полинома. Необходимо упростить ее и сохранить ее функциональность.

Полный разбор данной задачи с проведением свертки и решением по АРИЗ приведен в приложении 3.

Формулировка задачи не предполагает сама по себе какого бы то ни было противоречия. Тем не менее, быстро найти подход к ее решению также не удается. После проведения свертывания системы была получена функционально-идеальная модель и сформулирована задача свертывания. После этого противоречие было легко локализовано и была поставлена изобретательская задача, решение которой по АРИЗ’у привело к контрольному решению.

Таким образом было показано, что выбранный алгоритм постановки изобретательских задач применим для задач в программировании.

2.3. Описание основного алгоритма

Для написание программного продукта требовался единый алгоритм, который бы позволял объединить выбранные методы анализа системы и постановки задачи. С этой целью был проведен подробный разбор практического примера и на его основе составлен основной алгоритм программы.

Основным алгоритмом является алгоритм свертки. Он включает в себя ряд более простых алгоритмов, направленных на сбор и формирование вспомогательной информации. Их выполнение не является полностью линейными, так как на некоторых этапах они допускают возможность распараллеливания, и некоторые из них подразумевают циклическое повторение. Для адаптации было принято решение отказаться от параллелизма. Поэтому для программной реализации основной алгоритм был разбит на девять шагов, выстроенных по порядку, согласно их логике, два из которых заключено в цикл. Блок-схема основного алгоритма представлена в приложении 1.

Шаг 1. Построить компонентную модель

Этот этап представляет собой реализацию построения компонентной модели описываемой системы. Происходит сбор общей информации. Для этого производится ввод названия системы, определяется главная функция системы, определяются все компоненты самой системы, а также внесистемные компоненты, взаимодейтсвующие с ней.

Выходными данными шага 1 служит общая (определены только компоненты) модель рассматриваемой системы.

Шаг 2. Построить структурную модель

На данном шаге происходит добавление в нашу модель связей между системными, внесистемными компонентами и их связей друг с другом. Для

этого строится матрица взаимодействия компонентов *М*. При этом считается, что компонент *а* взаимодействует с компонентом *б*, если *М*(*а,б*) = “+” и не взаимодействует, если *М*(*а,б*) = “-”. Матрица строится заполненной “-”, при этом элементы на главной диагонали не могут быть изменены. Это связано с тем, что для всякого компонента всегда существует возможность взаимодействия с самим собой. Матрица подразумевает направленный характер взаимодействия. Хотя, на первый взгляд, она должна быть симметричной, это не является обязательным условием.

Выходными данными шага 2 является модель системы с определенными взаимодействиями между системными и внесистемными компонентами.

Шаг 3. Построить функциональную модель

Этот шаг служит для спецификации взаимодействий компонентов. Если два компонента взаимодействуют между собой, то существует некоторая функция, объектом которой служит один, а субъектом – другой компоненты. Таким образом, задать функцию возможно лишь между теми компонентами, между которыми было указано наличие взаимодействия на предыдущем шаге. Определение функции может быть получено либо из текстового названия, либо из комбинации названия параметра и направления его изменения. Всякая функция характеризуется набором дополнительных параметров, который включает ранг функции (вспомогательная 1 - 5, основная 1 - 3, вредная) и уровень выполнения (избыточный, адекватный, недостаточный). При этом уровень выполнения вредной функции отсутствует.

Выходными данными шага 3 служит модель системы, содержащая компоненты и все их функции.

Шаг 4. Провести причинно-следственный анализ

Наиболее удобным и наглядным методом проведения причинно-следственного анализа является построение графа, который бы содержал три типа вершин – одну вершину являющуюся целевым нежелательным эффектом системы, неограниченное количество промежуточных вершин, описывающих причины появления нежелательного эффекта, и конечных вершин, каждая из которых представляет собой ключевой недостаток.

Перед построением графа задается ключевой недостаток. Дальнейшее построение причинно-следственного графа происходит в свободной форме. При этом для каждой конечной вершины формируется список компонентов системы, которых касается данный ключевой недостаток.

Выходные данные шага 4 отдельно не специфицируются, так как этот шаг непосредственно должен предшествовать со связанным с ним шагом 5. Для основного алгоритма значение имеют только выходные данные шага 5.

Шаг 5. Построить диагностическую таблицу

На этом этапе строится диагностическая таблица. Она строится автоматически на основе данных, полученных на шаге 4 после окончания построения причинно-следственного графа. Производится подсчет количества недостатков для каждого компонента системы, после чего они ранжируются в порядке убывания количества недостатков. Таким образом, для каждого компонента определяется порядок свертывания.

Выходными данными шага 5 являются строго определенная последовательность компонентов рассматриваемой системы.

Шаг 6. Построить граф системы

На данном шаге происходит построение графа, которые является визуальным представлением системы. Входными данными здесь служит

модель системы, полученная на выходе шага 3. Данный граф является ориентированным. Вершины графа представляют собой компоненты системы. Каждая дуга соответствует некоторой функции, причем началом дуги является вершина, представляющая компонент-субъект функции, а концом - вершина, представляющая компонент-объект.

Выходными данными шага 6 служит граф системы, отображающий компоненты системы и их функции.

Шаг 7. Выбрать правило свертывания

Шаг 7 является началом цикла, который реализует непосредственно процедуру свертывания. Согласно последовательности, полученной на шаге 5, определяется порядок свертывания. Для компонента с наивысшим порядком свертывания определяется одно из трех правил свертывания. Поскольку алгоритм по своей сути является итерационным, и при каждой итерации мы выбираем одну из трех возможностей, то результат его выполнения представляет собой путь в троичном дереве.

Выходными данными шага 7 являются компонент системы и номер правила его свертывания.

Шаг 8. Перестроить граф

После того, как для компонента было выбрано правило свертывания, оно применяется. При этом указанный компонент подвергается свертке, то есть удаляется из графа соответствующая вершина согласно определенному правилу.

Правило 1

Элемент можно удалить, если нет объекта функции.

Вершина графа, соответствующая текущему элементу удаляется. Также удаляются вершины, в которые из нее ведут дуги. Удаляются все инцидентные этим вершинам дуги.

Для модели системы это значит удаление текущего компонента, всех объектов его функций а также всех функций, субъектом которых является один из удаленных компонентов, и функций, объектом которых является текущий компонент.

Правило 2

Элемент можно удалить, если функцию выполняет сам объект функции.

Текущая вершина графа удаляется. Удаляются дуги, ведущие в нее. Исходящие из нее дуги переносятся на вершины, в которые они вели.

В модели системы удаляется текущий компонент и все функции, объектом которых он является. Функции, для которых он является субъектом, передаются своим объектам, то есть их субъект становится равен их объекту.

Правило 3

Элемент можно удалить, если функцию выполняют оставшиеся элементы системы.

Текущая вершина графа удаляется. Удаляются все инцидентные ей дуги.

В модели системы удаляется текущий компонент и все функции, объектом которых он является. Функции, для которых он является субъектом переносятся в специльный пул. Это список, содержащий функции, которые должны быть реализованы оставшимися элементами системы.

После перестройки графа, текущий компонент убирается из рассмотрения. Если для свертывания остались компоненты, то происходит возврат к шагу 7. Если компонентов, подлежащих свертыванию, не осталось, шаг 8 считается законченным.

Выходными данными шага 8 является граф системы и модель системы, которые прошли процедуру свертывания.

Шаг 9. Сформулировать задачу свертывания

Этот этап является заключительным в основном алгоритме. Граф системы теперь представляет собой функционально-идеальную модель системы. На ее основе формулируется задача свертывания, то есть задача, возникшая в результате выполнения алгоритма. Также формулируется противоречие, которое необходимо разрешить.

Выходными данными служат формулировка задачи свертывания и формулировка противоречия.

Основной алгоритм разбит на 9 шагов. Входными данными служит описание системы на естественном языке. Выходными данными являются формулировка задачи свертывания и формулировка противоречия на естественном языке.

Глава 3. Описание программного продукта

После того, как была обоснована применимость алгоритма решения изобретательских задач в программировании, можно перейти ко второму этапу данной работы. А именно, к созданию программного продукта, реализующего алгоритмы постановки и анализа изобретательских задач.

Совместно со специалистами компании ЦИТК “Алгоритм” был выбран алгоритм, который был положен в основу программного продукта. Им стал

алгоритм “свертки”, который позволяет локализовать и обострить противоречие в изобретательской задаче. Ранее было показано, что он применим для некоторых задач из области программирования.

3.1. Архитектура системы

При разработке любого сложного программного продукта необходимо уделять достаточно внимания его архитектуре. Причем она требует внимания тем больше, чем большее количество функций планируется реализовать, чем более сложный алгоритм лежит в основе. Проектирование архитектуры любой системы является одним из важнейших и, зачастую, самых сложных этапов создания. Она как фундамент, влияет на все, что будет делаться после. Поэтому именно от ее правильного определения зависят сроки создания, надежность, удобство модернизации и даже удобство эксплуатации.

Хотя, на первый взгляд, описанный выше основной алгоритм является линейным и может показаться довольно простым, при его реализации возникает ряд трудностей, решить которые возможно во многом только при помощи правильной архитектуры приложения.

Эти трудности связаны, во-первых, с расхождением восприятия этого алгоритма машиной и человеком представлениях. Хотя с точки зрения входных-выходных данных, алгоритм не предполагает какой-либо сложной ветвистой структуры или параллелизма, человек воспринимает его не полностью линейно. При работе, когда в ходе выполнения возникает ошибка, компьютер может начать выполнять алгоритм заново. Для человека же это не является стандартным поведением. Вместо отмены всего совершенного прогресса, более разумным будет предоставить возможность отмены лишь части оного. То есть предоставить возможность возврата к месту совершения ошибки и ее исправлению. При таком возврате нарушается достоверность всех выходных данных шагов, входные данные которых окажутся

модифицированы после исправления ошибки. Это требует внимательного контроля целостности и своевременного пересчета устаревшей информации.

Во-вторых, данный алгоритм в своем изначальном виде предполагает выполнения некоторых вспомогательных действий, которые могут быть простыми при совершении человеком, но представлять определенные трудности для программной реализации. Например, построение графа с помощью листа бумаги и ручки не вызовет никаких трудностей. Однако автоматическое создание и представление такого же графа программными средствами будет представлять нетривиальную и трудоемкую задачу.

Учитывая эти аспекты, особенно важно было создать такую архитектуру системы, чтобы она позволяла легко проводить модернизацию, быть гибкой и в то же время довольно сильно формализованной.

Архитектура системы состоит из трех главных элементов: компонентной модели предметной области, уровня бизнес-логики и уровня логики пользовательского интерфейса. Было принято решение разделить логику уровня интерфейса от логики бизнес-уровня путем создания объектной модели. Предпосылками к этому послужила необходимость создания богатого пользовательского интерфейса и скрытия некоторых вспомогательных процессов от пользователя. Объектная модель предметной области позволила не только обеспечить необходимый уровень формализации, но и упростить задачу контроля потоков данных внутри приложения, а также обеспечить гибкость системы.

Для любого элемента предметной области – функции, компонента или, например, матрицы взаимодействия - создается описывающий его класс, который инкапсулирует все необходимые параметры. Эти параметры являются внутренними членами класса. Так как язык программирования Java не поддерживает свойства в классическом их понимании (в виде интерфейса доступа к переменной объекта), то для каждого значения создается пара из

акцессора и мутатора. Такой подход позволяет предотвратить ненужный доступ к членам классов, который может привести к несанкционированному изменению состояний объектов. Определение же у каждого класса необходимых методов позволяет, с одной стороны, строго формализовать взаимодействия и, с другой, сделать классы независимыми от структуры друг друга.

Одним из самых важных объектов является модель системы, описываемая специльным классом SystemModel. Данный класс содержит списки компонентов системы, внесистемных компонентов, а так же общую информацию о системе, такую как ее название и главная функция. На примере этого класса можно хорошо проиллюстрировать все преимущества выбранного подхода. Метод-акцессор, выдающий все компоненты системы, не зависит от того, что представляет собой компонент сам по себе. Но при этом позволяет контролировать целостность модели системы, предотвращая появление в качестве ее компонентов нежелательных данных.

Объект класса SystemModel создается на первом шаге алгоритма и является его выходными данными. Как отмечалось ранее, одной из основных проблем при разработке является различие восприятия основного алгоритма человеком от его интерпретации компьютером. Данный алгоритм при своем естественном использовании не ограничивает пользователя в возможности внесения изменений или корректировке ошибок на более ранних стадиях. Однако его адаптация под программную реализацию является, по большей части, линейной. Чтобы появилась возможность возврата на предыдущие шаги, алгоритм пришлось бы неоправданно усложнять, добавляя в него большое количество циклов и условий их выполнения. Определенная выше архитектура позволила решить эту проблему, проявив свои сильные стороны. Когда возникает проблема с возвратом по алгоритму, наибольшую трудность представляет соблюдение корректности и достоверности выходных данных шага, на котором были произведены изменения. Так как эти данные могут

быть использованы на различных последующих этапах, то, на первый взгляд, пересчет всех последующих шагов является единственной возможностью гарантировать правильность работы всего алгоритма. Тем не менее, такой пересчет требовал бы не только значительного дополнительного времени вычислений, но и доставлял бы пользователю продукта неоправданные трудности, связанные с повторным введением неизменившихся данных на последующих шагах.

Благодаря выделенной объектной модели удалось найти оптимальное решение этой конфликтной ситуации. Модель предметной области содержит все объекты, являющиеся выходными данными шагов основного алгоритма. Вместо того чтобы на каждом шаге создавать в качестве результата новый объект, было принято решение изменять уже существующий. Была создана так называемая “объектная подложка” основного адгоритма. Таким образом, на вход каждого шага в дополнение к необходимым данным подается уже готовый объект, который будет служить требуемым объектом выходных данных. Каждый шаг алгоритма теперь предусматривает один из двух вариантов прохождения, различия между которыми сведены к минимуму.

1. Если шаг выполняется в первый раз, объекта создается пустым, а конструктор инициализирует все необходимые поля. В этом случае сам шаг ничем не отличается от выполнения оного без объектной подложки.

2. Если шаг выполняется повторно, то есть после возможного внесения изменений на одном или нескольких предыдущих этапах, то объект выходных данных уже существует. Возможно, вследствии изменений, его состояние будет отличаться от того, каким оно было при первом завершении работы этого шага. Чтобы обеспечить соблюдение корректности необходимо использовать ту информацию об изменениях, которая оказывается инкапсулирована во входном объекте.

Рассмотрим работу этого механизма на примере шага 1 и шага 3. Выходными данными шага три является модель системы с заданными компонентами и всеми функциями каждого компонента. После завершения шага 3 пользователь решает внести изменения в процесс выполнения алгоритма. Например, возникает необходимость удалить один из компонентов системы. Когда происходит возврат к первому шагу и удаление ненужного компонента, соответствующие изменения происходят и в объектной модели. Объект класса SystemModel перестает содержать удаленный компонент. При этом целостность данных контролируется поведением объекта модели системы. То есть в данном случае проверяется, что у других компонентов отсутствуют функции, объектом которых является удаленный компонент. В случае необходимости, происходит автоматическое удаления заведомо ложных функций. Когда выполнение основного алгоритма вновь доходит до шага 3, то объект модели добавляется к входным данным. Так как на выходе шага модель системы должна содержать все функции своих компонентов, то необходимо узнать, какие изменения были внесены. Разность между множеством функций компонентов после первого выполнения и множеством функций, полученного из входного объекта модели системы и есть все произведенные и значимые для данного шага изменения. При контроле целостности на шаге 1 эти изменения были соответствующим образом учтены, когда происходило удаление некорректных функций. Поэтому все, что остается сделать на шаге 3 – это использовать функции из входного объекта в качестве заданных изначально.

Еще одной важной частью архитектуры является логика уровня интерфейса. Создания удобного и в то же время богатого пользовательского интерфейса привело к необходимости выделения логики его работы в отдельный уровень. Как было отмечено ранее, при программной реализации возникает ряд дополнительных процессов, которые связаны с выполнением основного алгоритма. Эти процессы не являются явными для пользователя,

поэтому их сокрытие является важной задачей. Ответственность за выполнение таких процессов была передана на уровень бизнес-логики, а логика пользовательского интерфейса была сохранена максимально близкой к логике выполнения основного алгоритма свертки.

Бизнес-логика программного продукта обычно представляет собой совокупность правил, принципов, зависимостей поведения объектов предметной области (области человеческой деятельности, которую система поддерживает). В данной архитектуре она служит не только для реализации правил и ограничений автоматизируемого алгоритма, но и для связи машинного и человеческого его представлений. С одной стороны, уровень бизнес-логики взаимодействует с объектной моделью. При этом происходит управление состояниями объектов, обеспечивается реализация всех необходимых для этого процессов, контролируется корректность работы основного алгоритма. С другой стороны, бизнес-логика взаимодействует с уровнем логики интерфейса, который представляет собой тот же основной алгоритм, но в привычной для человеческого восприятия форме.

Такое разделение всей логики приложения на два уровня обеспечило простоту создания интерфейса и гарантировало его гибкость.

3.2. Пользовательский интерфейс системы

Как отмечалось ранее, восприятие алгоритма свертки человеком отличается от его программной реализации. Создание продуманной архитектуры приложения позволило нивелировать эти отличия в части, касающейся программирования алгоритма. Однако, не менее важно было гарантировать удобство использования программного продукта человеком.

При создании пользовательского интерфейса ставилось две задачи. Поскольку программный продукт представлял бы собой автоматизацию уже существующего и используемого алгоритма, то особенно важно было максимально сохранить единообразие выполнения шагов вручную и

автоматически через приложение. Создание интерфейса, не отличающегося от того, с которым привык иметь дело пользователь, с тем, который получается при помощи листа бумаги и ручки, позволило бы обеспечить удобство использования программного продукта.

Второй задачей ставилось свести количество ошибок при выполнении алгоритма к минимуму. Частично, эта задача решалась созданием хорошей архитектуры приложения. Также, факт приближенности интерфейса программной реализации алгоритма к интерфейсу традиционного его использования позволял сократить неточности и расхождения в процессе выполнения, что естественным образом снижало риск возникновения ошибок. Но основное решение состояло в создании такого интерфейса, который бы, предоставлял пользователю достаточную свободу и ограничивал бы вредоносные или неверные его действия.

Для решения обеих задач специалистами компании ЦИТК ”Алгоритм” был предоставлен образец натурального выполнения алгоритма свертки, после чего он подвергся тщательному анализу. В ходе анализа были выделены этапы работы алгоритма, которые легли в основу пользовательского интерфейса, и определены наиболее важные для переноса аналогии.

В ходе анализа в традиционном использовании алгоритма было выявлено два критически важных момента, требовавших максимально точного переноса на пользовательский интерфейс программного продукта.

Первый был связан с необходимостью использования графов при выполнении некоторых шагов алгоритма. Эта проблема касалась в первую очередь графического представления. Если программное создание модели графа было задачей тривиальной, то его отображение могло потребовать написания объемного и сложного кода. К тому же, требовалось не просто статически оторажать определенный граф, но и проводить с ним сложные

операции, динамически перестраивать его структуру в ходе работы алгоритма, обеспечивать его интерактивное взаимодействие с пользователем. Решение этой задачи было довольно сложным, оно требовало на свою реализацию много времени и, к тому же, не представляло научного интереса для данной работы. Поэтому было принято решение воспользоваться внешними библиотеками, поддерживающими работу с графами. Найденные библиотеки потребовали некоторой адаптация под специфику конкретного алгоритма, но со своей задачей справились полностью. В результате в пользовательском интерфейсе в полном объеме были реализованы все требуемые для работы основного алгоритма взаимодействия с графическими представлениями графов. Результат можно увидеть на рис. 1, где показано представление графа причинно-следственного анализа.

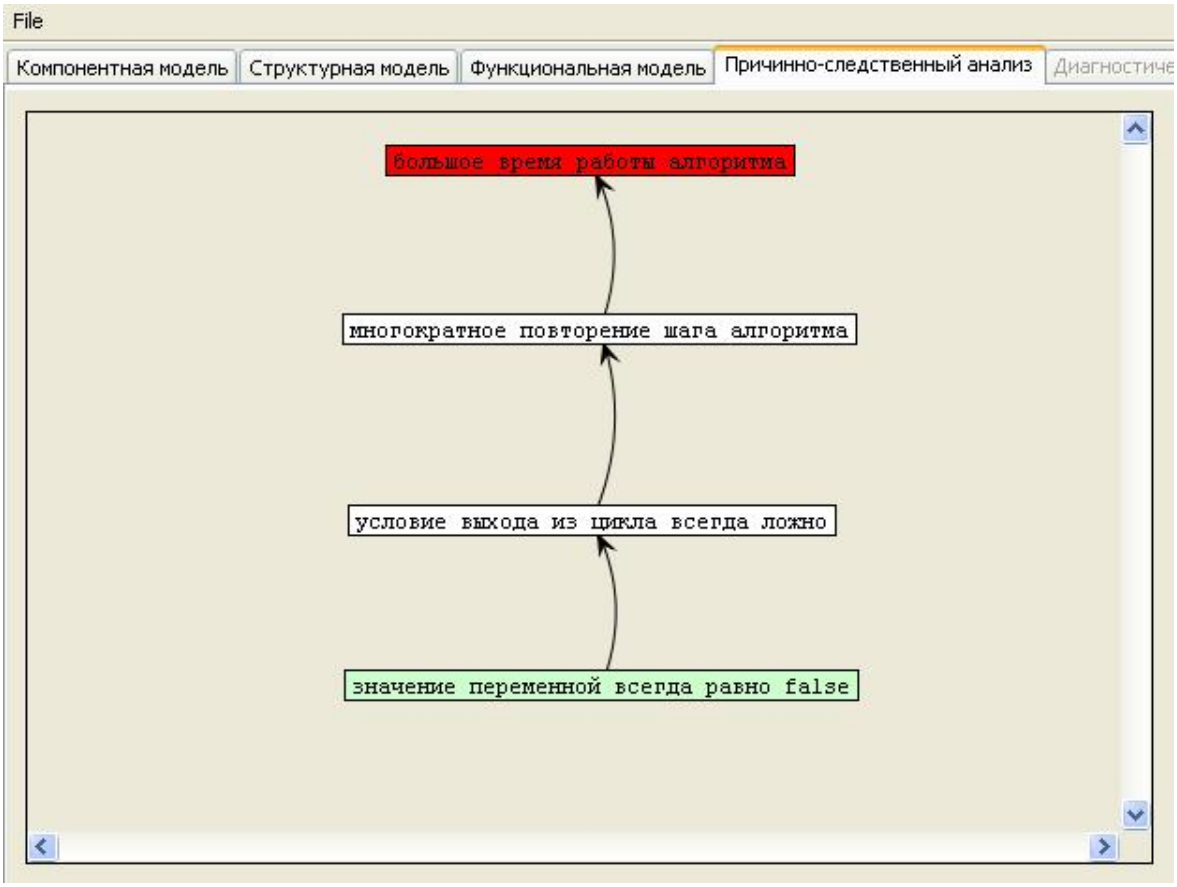


Рис. 1

Второй критически важный момент был связан с многообразием вводимой информации. Основной алгоритм в процессе своего выполнения

должен работать с большими объемами информации. Если для логики приложения форма представления этой информации не имеет сущетсвенного значения, то для пользователя форма представляемой информации может быть даже первична содержанию. Так как человек привык работать с информацией не в чистом виде, как то делает компьютер, а через различные формы-посредники, то если не предоставить ему такой возможности, это может повлечь существенное замедление работы пользователя и увеличение количества допускаемых им ошибок. Алгоритм свертки требует ввода большого количества данных. Сам ввод также связан с анализом пользователем информации, предоставляемой программой. Поэтому было необходимо использовать те же формы представления информации, которые использует человек при работе с алгоритмом вручную.

Центральной формой представления здесь служит таблица в самом широком понимании. Предоставляемый языком Java инструментарий для создания таблиц оказался недостаточен для реализации полной аналогии пользовательского интерфейса с требуемым. Было создано расширение, которое позволило максимально полно сблизить процессы выполнения алгоритма в двух его различных вариантах. Созданные в результате формы представления информации позволили систематизировать ее отображение, снизить когнитивную нагрузку на пользователя и ускорить его работу. Пример разработанной формы можно увидеть на рис. 2. Графическое представление матрицы взаимодействия на шаге 2 позволяет пользователю видеть модель системы целиком и заполнять ее на основе имеющихся данных так же легко, как он делал бы это при выполнении алгоритма самостоятельно.

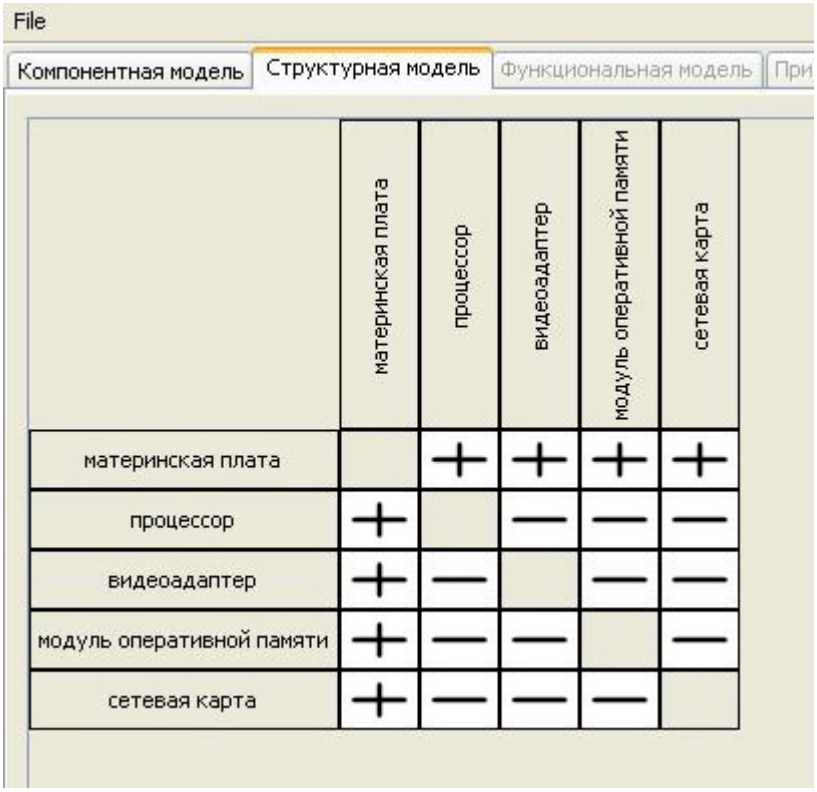


Рис. 2

3.3. Технологии и инструменты

Для создания системы был выбран язык программирования Java. Он является объектно-ориентированным языком, поэтому позволяет в полной мере реализовать задуманную архитектуру программного продукта. Также немало важен и тот факт, что Java является одним из самых распространенных языков программирования, вследствии чего существует множество сторонних билиотек, расширяющих его функциональность. Это послужило второй причиной выбора данного языка, так как для полной реализации идеи пользовательского интерфейса была необходима внешняя библиотека для работы с графами.

Необходимость создания сложного пользовательского интерфейса определила выбор среды разработки, в качестве которой выступила NetBeans IDE. Она является средой с открытым кодом, к ее преимуществам относятся широкая функциональность и поддержка всех современных технологий. Кроме того, NetBeans предлагает разработчику механизм разделения

приложения на интерфейс и логику. Такое разделения естественным образом соответстсвует архитектуре создаваемого приложения.

Для написания пользовательского интерфейса используются две библиотеки - AWT и Swing. Последняя предоставляет набор гибких интерфейсных компонент и отвечает идеологии языка Java - то есть обеспечивает кросс-платформенность написанных на ней приложений. Кроме того Swing поддерживает разделение компонент на две части: то, как компонент выглядит, и то, как он себя ведет. Это дает возможность разграничить логику компонента от его вида, что полностью отвечает требованиям разрабатываемой архитектуры.

Для представления графов была взята библиотека с открытым кодом JUNG (the Java Universal Network/Graph Framework). Она не только имеет в своей основе надежную архитектуру, позволяющую работать с любым вариантом представления графа, но и включает мощный каркас для визуализации графов и определению пользовательского взаимодействия с ними.

3.4. Преимущества разрабатываемого программного продукта

Как отмечалось выше, программная реализация алгоритма свертки преследует несколько целей. Она должна быть достаточно мощной и надежной, стабильной в работе, чтобы эффективно поддерживать выполнение алгоритма. Кроме того, ее пользовательский интерфейс должен сочетать в себе такие противоречивые качества, как мощность и функциональность, с удобством и естественностью использования. Совокупность этих факторов делает данный программный продукт уникальным.

Он позволяет автоматизировать вспомогательные процессы при выполнении алгоритма свертки, минимизировать затрачиваемые на

свертывание системы пользователем усилия и свести риск возникновения ошибок технического характера к минимуму. Подход, при котором пользовательский интерфейс в полной мере соответствует естественному, позволяет упростить и удешевить процесс внедрения данного продукта.

Круг потенциальных пользователей вследствие этого также очень широк. Вследствие автоматизации сопутствующих алгоритму процедур, программный продукт может применяться не только для решения производственных задач специалистами ТРИЗ, но и при обучении этой методики, проведении семинаров в качестве наглядного материала или проведении практических работ. Сочетание строгого соблюдения алгоритма свертывания и созданного пользовательского интерфейса обеспечивает эффективность использования продукта при решении реальных задач и простоту, наглядность при обучении ТРИЗ.

Заключение

В данной работе был разработан инструмент, способный разрешать возникающие в программировании противоречия. Показано, что АРИЗ является эффективным методом решения изобретательских задач в программировании.

Были решены задачи обоснования применимости методов ТРИЗ для решения изобретательских задач в программировании. На основе конкретных примеров было доказано, что современный вариант АРИЗ позволяет решать учебные изобретательские задачи различной сложности не только в технической области, но и в области программирования.

В данной работе также был проведен анализ вспомогательных алгоритмов, призванных обеспечить корректность постановки и формулировки изобретательской задачи. Современный характер программирования предполагает возможность появления широкого круга

задач, формализация которых может представлять серьезные трудности, но являться необходимой для получения решения по АРИЗ.

Были решены задачи исследования и адаптирования алгоритмов постановки изобретательских задач. В результате разработан программный продукт, реализующий ряд алгоритмов постановки и анализа изобретательских задач. В основу программы лег алгоритм свертки, который был адаптирован под программную реализацию. Получившийся продукт позволяет автоматизировать выполнение этого алгоритма. Он обладает продуманной архитектурой, которая делает выполнение алгоритма столь же естественным, как и в естественной форме. Богатый пользовательский интерфейс, созданный так, чтобы максимально приблизить процесс взаимодействия с программой к выполнению алгоритма “на бумаге”, уменьшает когнитивную нагрузку на пользователя. Таким образом, разработанный программный продукт может быть легко использован как специалистами ТРИЗ для решения сложных изобретательских задач, так и программистами, желающими улучшить результаты своей работы. Немаловажным является и тот факт, что приложение, в силу простоты работы с ним, может быть использовано при обучении ТРИЗ.

Были достигнуты обе цели:

1. Доказано, что методы ТРИЗ можно эффективно применять для постановки и решения изобретательских задач в программировании.

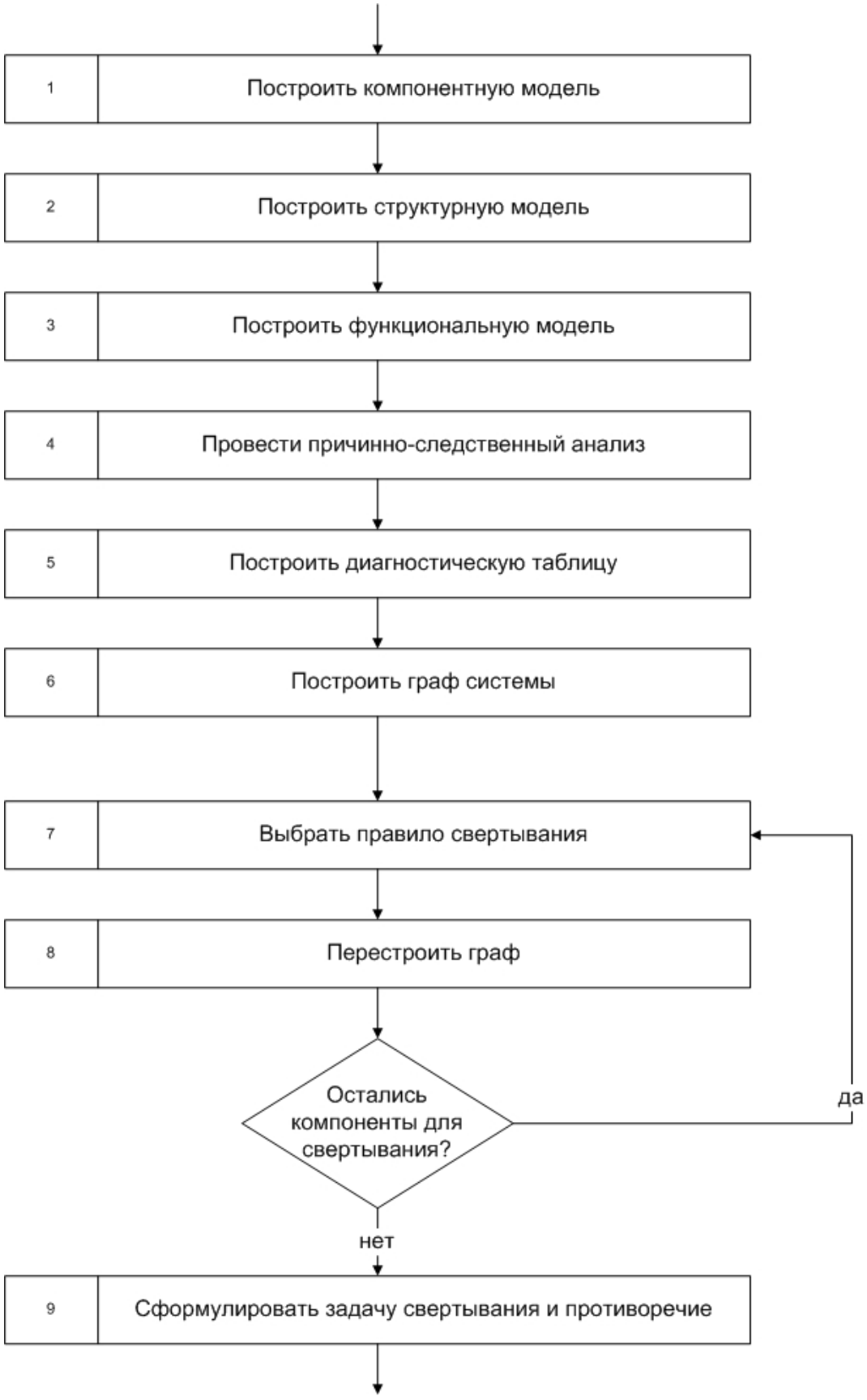
2. Разработан программный продукт, основанный на алгоритмах постановки изобретательских задач, который автоматизирует вспомогательные процессы и облегчает выполнение этих алгоритмов.

Список литературы

1. John Stamey. TRIZ and Extreme Programming. Proceedings of Southeast InfORMS, October 2006. [Электронный ресурс] http://www.triz.org.uk/papers/TRIZandExtremeProgramming.pdf
2. ТРИЗ: Устранение программных противоречий – Философия программирования – RSDN [Электронный ресурс] : форум программистов RSDN http://www.rsdn.ru/forum/philosophy/3277689.all.aspx
3. Сычев С.В., Лебедев К.А. Как вспомнить "и так известное". [Электронный ресурс] http://www.triz-ri.ru/themes/method/creative/creative50.asp
4. Одинцов И.О., Рубин М.С. Повышение эффективности разработки программных продуктов на основе методов ТРИЗ // Научно-практическая конференция «ТРИЗ-Фест 2009»: сборник трудов конференции. СПб., 2009. [Электронный ресурс] http://www.trizsummit.ru/file.php/id/f4434/name/Одинцов-TRIZ-SW.doc
5. Официальный сайт компании Метод. [Электронный ресурс] http://method.ru/
6. Официальный сайт компании Invention Machine. [Электронный ресурс] http://www.invention-machine.com/
7. Глоссарий к основанной на базе ТРИЗ методике G3:ID. [Электронный ресурс] http://gen3.ru/3605/5453/#_ftn1
8. Альтшуллер Г.С. АРИЗ - значит победа. Алгоритм решения изобретательских задач АРИЗ-85-В. // Правила игры без правил // Сост. А.Б.Селюцкий. Петрозаводск: Карелия, 1989.
9. Альтшуллер Г.С. Творчество как точная наука. Теория решения изобретательских задач. - М.: Сов. радио, 1979.-184 с. - Кибернетика.
10. Альтшуллер Г.С. История развития АРИЗ (конспект). Баку, 1986 http://www.trizscientific.com/TRIZ_sci/history/gsa_hist_devel_ariz88_r.htm

11. Petrov V. TRIZ – past, present and future. Keynote speech. // ETRIA World Conference - TRIZ Future 2003. November 12-14, 2003.
12. Петров В.М., Рубин М.С. АРИЗ-85-В - недостатки и предложения по развитию. [Электронный ресурс] http://www.triz-summit.ru/ru/section.php?docId=4189
13. Рубин М.С. Об АРИЗ нового поколения: многоаспектный цикл преодоления противоречий. Научно-практическая конференция «ТРИЗ-ФЕСТ 2009»: сборник трудов конференции. СПб, 2009. – 302 с. http://www.triz-summit.ru/file.php/id/f4365/name/АРИЗ-2010-Фест-PMC.doc
14. Рубин М.С. Функционально-логическая блок-схема для анализа работы алгоритма. Предложения по изменению структуры и текста АРИЗ-85-В. [Электронный ресурс] http://www.trizsummit.ru/ru/section.php?docId=4176
15. Герасимов В.М., Дубров В. Е.; Карпунин М. Г., Кузьмин А. М., Литвин С. С. Применение методов технического творчества при проведении функционально-стоимостного анализа: Методические рекомендации. М.: "Информэлектро", 1990, 60 с.

Приложение 1. Блок-схема основного алгоритма



Приложение 2. Разбор задач по АРИЗ

2.1. Задача о сортировке массива

Массив, например содержащий целые числа, можно отсортировать методом пузырька. Однако время выполнения алгоритма растет квадратично длине массива, то есть для достаточно длинных массивов это время становится недопустимо большим. Можно создать новый алгоритм сортировки, но на это также потребуется много времени. Создание нового метода “с нуля” является трудной задачей.

Как сократить время сортировки длинного массива, не создавая новый алгоритм сортировки?

ШАГ 1

1. Сформулировать главное требование к системе, целевую характеристику.

Необходимо уменьшить время сортировки массива, не создавая новый алгоритм сортировки (используя существующий алгоритм сортировки).

ШАГ 2

2.1. Сформулировать главные функции, которые необходимо выполнить: носитель функции (субъект), действие и/или изменение параметра, объект функции и имеющиеся ограничения.

носитель функции (может быть не известен по условиям задачи): неизвестен

действие и/или изменение параметра (обязательно): уменьшить время работы

объект функции (обязательно): алгоритм сортировки

имеющиеся ограничения: нужно использовать существующий алгоритм сортировки

Уменьшить время сортировки массива целых чисел, используя существующий алгоритм (не затрачивая большое количество времени на создание нового алгоритма сортировки).

2.2.1. Функциональный ИКР. Объект (назвать) САМ делает (описать) в период (назвать) при обязательных условиях (описать).

Массив САМ уменьшает время сортировки во время работы алгоритма при использовании существующего алгоритма сортировки.

2.2.2. ФОП.

пропущен

ШАГ 3

3.1. Противоречие требований (ПрТ): ЕСЛИ ...(указать вносимое изменение).., ТО (+ указать главное требование), НО (–указать нежелательное требование).

ЕСЛИ создается новый алгоритм, ТО уменьшается время сортировки массива, НО на создание нового алгоритма уйдет слишком много времени.

ЕСЛИ используется существующий алгоритм, ТО массив сортируется, НО сортировка занимает слишком много времени.

3.2.1. Таблица применения типовые приемов разрешения противоречий требований (технических противоречий).

Будем использовать сокращенную таблицу.

Нужно изменить: Скорость (9)

Ухудшается: Удобство изготовления (32), Сложность устройства (36)

По таблице получаем: 9/32 - (13,1), 9/36 - (10,4,34)

13. ПРИНЦИП "НАОБОРОТ".

а. Вместо действия, диктуемого условиями задачи, осуществить обратное действие.

б. Сделать движущуюся часть объекта или внешней среды неподвижной, а неподвижную - движущейся.

в. Перевернуть объект "вверх ногами", вывернуть его.

1. ПРИНЦИП ДРОБЛЕНИЯ.

а. Разделить объект на независимые части.

б. Выполнить объект разборным.

в. Увеличить степень дробления объекта.

10. ПРИНЦИП ПРЕДВАРИТЕЛЬНОГО ДЕЙСТВИЯ.

а. Заранее выполнить требуемое действие (полностью или хотя бы частично).

б. Заранее расставить объекты так, чтобы они могли вступить в действие без затрат времени на доставку и с наиболее удобного места.

4. ПРИНЦИП АСИММЕТРИИ.

а. Перейти от симметричной формы объекта к асимметричной.

б. Если объект уже асимметричен, увеличить степень асимметрии.

34. ПРИНЦИП ОТБРОСА И РЕГЕНЕРАЦИИ ЧАСТЕЙ.

а. Выполнившая свое назначение или ставшая ненужной часть объекта должна быть отброшена (растворена, испарена и т.д.) или видоизменена непосредственно в ходе работы.

б. Расходуемые части объекта должны быть восстановлены непосредственно в ходе работы.

ШАГ 4

4.1.1. Конфликтующие элементы: не менее двух конфликтующих элементов, схема конфликта. Какой элемент можно менять?

Массив целых чисел

Алгоритм сортировки

Они взаимодействуют между собой, можно ограниченно менять массив целых чисел.

4.1.2. Сформулировать ОВ и ОЗ.

ОВ – время сортировки массива

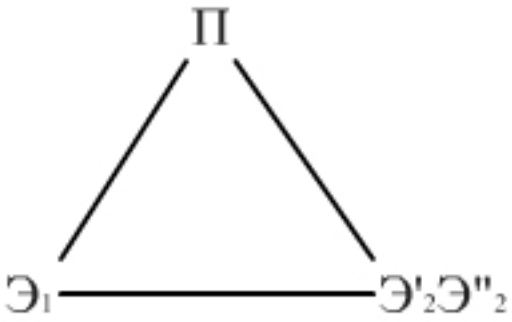
ОЗ – взаимодействие алгоритма сортировки с массивом целых чисел

4.1.3. Перечень основных ресурсов системы.

Массив целых чисел

4.2.1. Элепольное решение.

Э1 – алгоритм сортировки, Э’2,Э”2 – массивы целых чисел (принцип дробления, разрушение вредной связи между алгоритмом и массивом). Так как размер массива отрицательно влияет на время работы алгоритма, то между ними необходимо ввести некоторое поле П – правила взаимодействия.



4.2.2. ИКР1: Икс-элемент из ресурсов системы САМ УСТРАНЯЕТ (-), СОХРАНЯЯ (+)

Икс-элемент из ресурсов системы САМ УСТРАНЯЕТ большое время сортировки, СОХРАНЯЯ использование существующего алгоритма сортировки.

ШАГ 5

5.1. ПрС. Конфликтующий элемент должен обладать свойством Х, чтобы обеспечить главное требование (+), и должен обладать свойством "АНТИ-Х", чтобы устранить недостаток (-).

Конфликтующий элемент должен быть малого размера, чтобы обеспечить малое время выполнения алгоритма сортировки, и должен быть большим, чтобы хранить все необходимые числа.

5.2. ИКР2: Оперативная зона (указать) в течение оперативного времени (указать) должна САМА обеспечивать (указать противоположные макро- или микросостояния).

Взаимодействие алгоритма сортировки с массивом целых чисел в течение сортировки должно САМО обеспечивать малое время сортировки.

ШАГ 6

6.1 Мобилизовать ресурсы. Использовать ресурсы для разрешения конфликта.

В качестве ресурсов можно использовать: размер массива

6.2. Использовать информационные фонды: указатели эффектов, ФОП, линии развития в стандартах, законы развития.

пропущен

ШАГ 7

7. Переформулировка (уточнение) задачи.

7.1. если имеется идея (идеи) решения, то необходимо обязательно сформулировать ее (их) недостатки, проверить соответствие полученного решения формулировкам ИКР, приемам и стандартам и далее:

- либо оценить эти недостатки и не точности как не влияющие на работоспособность идеи,

- либо вернуться к рекомендациям по использованию приемов и стандартов,

- либо уточнить ресурсы для поиска решения,

- либо отказаться от предлагаемой идеи и вернуться к шагу1.

Решение

Согласно принципу дробления и принципу предварительного действия, нужно до начала сортировки разбить массив на части (например на две равные части). При этом каждый из двух новых массивов будет меньше начального.

Затем нужно применить алгоритм сортировки к каждому масссиву по отдельности, после чего соединить отдельные упорядоченные массивы в один. При этом так как время соединения упорядоченных массивов линейно зависит от их длины, а время сортировки пузырьком - квадратично, то общее время сортировки начального массива будет уменьшено. Причем чем длиннее массив, тем больший выигрыш во времени будет получен.

7.2. Если идея кажется удовлетворительной, но не ясно как ее исполнить, то необходимо перейти к формулировке вторичной задачи:

- уточнить целевую (искомую) функцию (объединить функции, перейти к подфункции и др.),

- уточнить или сделать противоположными ограничения,

- перейти к рассмотрению подсистем и сформулировать задачи для них.

7.3. Если найденное решение кажется удовлетворительным и работоспособным, то:

- рассмотреть последствия использования (реализации) предлагаемой идеи,

- рассмотреть задачу с позиции сопутствующих аспектов.

7.4. Остановить анализ, если новая задача не поставлена и нет ее изменений. В противном случае перейти к шагу 1.

2.2. Задача о режимах редактора геометрических фигур

В редакторе геометрических фигур (в основном – ломаных линий и многоугольников) существуют несколько режимов редактирования. Два из них отвечают за изменение формы редактируемых фигур. Первый режим – режим перемещения точек – позволяет с помощью мыши передвинуть любую вершину фигуры в заданную точку. Второй режим позволяет добавить к контуру новую вершину (щелчком мыши ребро разбивается на две части). При использовании такого редактора возникает неудобство: приходится постоянно переключаться из одного режима в другой.

Как сделать использование этих двух режимов удобным?

ШАГ 1

1. Сформулировать главное требование к системе, целевую характеристику.

Необходимо сделать использование двух режимов редактирования удобным для пользователя

ШАГ 2

2.1. Сформулировать главные функции, которые необходимо выполнить: носитель функции (субъект), действие и/или изменение параметра, объект функции и имеющиеся ограничения.

носитель функции (может быть не известен по условиям задачи): окно редактора (поле редактирования фигур)

действие и/или изменение параметра (обязательно): изменять (выбирать один из двух)

объект функции (обязательно): режим редактирования фигур

имеющиеся ограничения: обеспечить максимальное удобство выбора режимов редактирования, уменьшить время выбора, сохранить функциональность (сохранить все режимы редактирования)

Обеспечить возможность переключения режимов редактирования фигур в окне редактора, гарантировав удобство и быстроту выбора режимов и сохранив все режимы.

2.2.1. Функциональный ИКР. Объект (назвать) САМ делает (описать) в период (назвать) при обязательных условиях (описать).

Режим редактирования САМ осуществляет переключение во время редактирования фигур, сохраняя все режимы и обеспечивая удобство и малое время выбора.

2.2.2. ФОП.

пропущен

ШАГ 3

3.1. Противоречие требований (ПрТ): ЕСЛИ ...(указать вносимое изменение).., ТО (+ указать главное требование), НО (–указать нежелательное требование).

ЕСЛИ сделать один режим редактирования, ТО его будет легко использовать, НО редактор потеряет функциональность.

ЕСЛИ сделать для каждого режима свой способ выбора (например кнопку), ТО редактор сохранит всю функциональность, НО переключение будет слишком долгим и неудобным.

3.2.1. Таблица применения типовые приемов разрешения противоречий требований (технических противоречий).

Будем использовать сокращенную таблицу.

Нужно изменить: Удобство эксплуатации(33)

Ухудшается: Адаптация/универсальность(35)

По таблице получаем: 33/35 – (15,34,1,16)

15. ПРИНЦИП ДИНАМИЧНОСТИ.

а. Характеристики объекта или внешней среды должны меняться так, чтобы быть оптимальными на каждом этапе работы.

б. Разделить объект на части, способные перемещаться относительно друг друга.

в. Если объект в целом неподвижен, сделать его подвижным, перемещающимся.

34. ПРИНЦИП ОТБРОСА И РЕГЕНЕРАЦИИ ЧАСТЕЙ.

а. Выполнившая свое назначение или ставшая ненужной часть объекта должна быть отброшена (растворена, испарена и т.д.) или видоизменена непосредственно в ходе работы.

б. Расходуемые части объекта должны быть восстановлены непосредственно в ходе работы.

1. ПРИНЦИП ДРОБЛЕНИЯ.

а. Разделить объект на независимые части.

б. Выполнить объект разборным.

в. Увеличить степень дробления объекта.

16. ПРИНЦИП ЧАСТИЧНОГО ИЛИ ИЗБЫТОЧНОГО ДЕЙСТВИЯ.

Если трудно получить 100% требуемого действия или эффекта, надо получить "чуть меньше" или "чуть больше" - задача при этом может существенно упроститься.

ШАГ 4

4.1.1. Конфликтующие элементы: не менее двух конфликтующих элементов, схема конфликта. Какой элемент можно менять?

Фигура

Курсор мыши

Они взаимодействуют между собой, можно ограниченно менять и то, и другое.

Курсор мыши выполняет с фигурой действия, зависящие от режима редактирования.

4.1.2. Сформулировать ОВ и ОЗ.

ОВ – время редактирования фигуры

ОЗ – поле редактирования, область фигуры

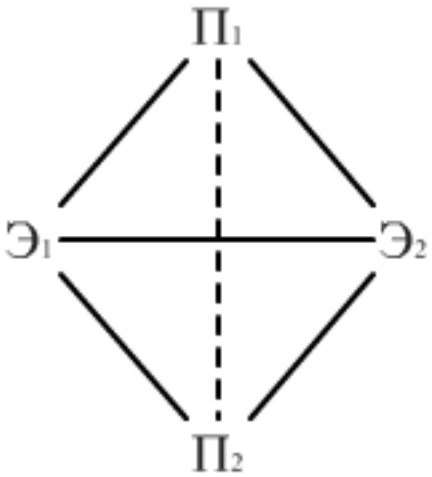
4.1.3. Перечень основных ресурсов системы.

Панели кнопок, фигура, курсор мыши

4.2.1. Элепольное решение.

Э1 – фигура, Э2 – курсор мыши, П1,П2 – два различных режима редактирования. Режимы подразумевают различное взаимодействие фигуры и курсора, при этом

активным может быть только один режим. Поэтому между П1 и П2 присутствует конфликт.



4.2.2. ИКР1: Икс-элемент из ресурсов системы САМ УСТРАНЯЕТ (-), СОХРАНЯЯ (+)

Икс-элемент из ресурсов системы САМ УСТРАНЯЕТ выбор режимов, СОХРАНЯЯ всю функциональность редактора (все режимы).

ШАГ 5

5.1. ПрС. Конфликтующий элемент должен обладать свойством Х, чтобы обеспечить главное требование (+), и должен обладать свойством "АНТИ-Х", чтобы устранить недостаток (-).

Конфликтующий элемент должен обладать свойством переключения, чтобы обеспечить выбор режимов редактирования и функциональность, и должен обладать свойством единственности, чтобы обеспечить удобство и скорость выбора.

5.2. ИКР2: Оперативная зона (указать) в течение оперативного времени (указать) должна САМА обеспечивать (указать противоположные макро- или микросостояния).

Область фигуры во время редактирования должна САМА обеспечивать возможность и удобство, простоту переключения режимов редактирования.

ШАГ 6

6.1 Мобилизовать ресурсы. Использовать ресурсы для разрешения конфликта.

В качестве ресурсов можно использовать: элементы поля редактирования, элементы окна редактора (например кнопки), части фигуры (вершины и ребра), форму курсора.

6.2. Использовать информационные фонды: указатели эффектов, ФОП, линии развития в стандартах, законы развития.

пропущен

ШАГ 7

7. Переформулировка (уточнение) задачи.

7.1. если имеется идея (идеи) решения, то необходимо обязательно сформулировать ее (их) недостатки, проверить соответствие полученного решения формулировкам ИКР, приемам и стандартам и далее:

- либо оценить эти недостатки и не точности как не влияющие на работоспособность идеи,

- либо вернуться к рекомендациям по использованию приемов и стандартов,

- либо уточнить ресурсы для поиска решения,

- либо отказаться от предлагаемой идеи и вернуться к шагу1.

Решение

Согласно пункту 5.2, область фигуры должна во время редактирования сама обеспечивать возможность выбора режимов. Используя принцип динамичности, характеристики объекта должны меняться так, чтобы быть оптимальными на каждом этапе. То есть режим должен меняться динамически во время редактирования, в зависимости от того, что пользователь собирается делать. По принципу дробления следует разбить область фигуры на зоны, в данном случае это зоны вершин и зоны ребер. Курсор мыши, при нахождении в определенной, зоне включает соответствующий режим редактирования и может, например, менять свою форму, чтобы пользователь мог определить, какой режим используется.

7.2. Если идея кажется удовлетворительной, но не ясно как ее исполнить, то необходимо перейти к формулировке вторичной задачи:

- уточнить целевую (искомую) функцию (объединить функции, перейти к подфункции и др.),

- уточнить или сделать противоположными ограничения,

- перейти к рассмотрению подсистем и сформулировать задачи для них.

7.3. Если найденное решение кажется удовлетворительным и работоспособным, то:

- рассмотреть последствия использования (реализации) предлагаемой идеи,

- рассмотреть задачу с позиции сопутствующих аспектов.

7.4. Остановить анализ, если новая задача не поставлена и нет ее изменений. В противном случае перейти к шагу 1.

2.3. Задача о защите общедоступной программы

Достаточно сложная и уникальная программа расчета была доступна сотрудникам института в виде файла в машинных кодах. Были опубликованы также результаты работы этой программы: исходные данные, результаты расчетов.

Как сделать так, чтобы доступной всем программой мог пользоваться только сам автор этой программы?

ШАГ 1

1. Сформулировать главное требование к системе, целевую характеристику.

Необходимо защитить программу от несанкционированного доступа, притом, что она остается доступной в виде файла в машинных кодах.

ШАГ 2

2.1. Сформулировать главные функции, которые необходимо выполнить: носитель функции (субъект), действие и/или изменение параметра, объект функции и имеющиеся ограничения.

носитель функции (может быть не известен по условиям задачи): неизвестен

действие и/или изменение параметра (обязательно): защитить

объект функции (обязательно): файл в машинных кодах

имеющиеся ограничения: программой может пользоваться только ее автор, файл в машинных кодах доступен всем

Защитить файл в машинных кодах от доступа любого, кроме ее автора, сохранив его доступность.

2.2.1. Функциональный ИКР. Объект (назвать) САМ делает (описать) в период (назвать) при обязательных условиях (описать).

Программа САМА в момент обращения к ней защищает себя от доступа всех, кроме ее автора, сохраняя свою доступность.

2.2.2. ФОП.

пропущен

ШАГ 3

3.1. Противоречие требований (ПрТ): ЕСЛИ ...(указать вносимое изменение)., ТО (+ указать главное требование), НО (−указать нежелательное требование).

ЕСЛИ мы убираем файл в машинных кодах из вычислительного центра, то мы защищаем ее от несанкционированного доступа, но теряем ее доступность (нарушаем правила работы центра).

ЕСЛИ мы оставляем файл в машинных кодах в вычислительном центре, то мы сохраняем его доступность, но не защищаем его от несанкционированного доступа.

3.2.1. Таблица применения типовые приемов разрешения противоречий требований (технических противоречий).

Будем использовать сокращенную таблицу.

Нужно изменить: Потери информации(24), Вредные факторы, действующие на объект(30)

Ухудшается: Удобство эксплуатации(33), Удобство изготовления(32)

По таблице получаем: 24/33 - (27,22), 30/32 - (24,2)

22. ПРИНЦИП "ОБРАТИТЬ ВРЕД В ПОЛЬЗУ".

а. Использовать вредные факторы (в частности? вредные воздействия среды) для получения положительного эффекта.

б. Устранить вредный фактор за счет сложения с другими вредными факторами.

в. Усилить вредный фактор до такой степени, чтобы он перестал быть вредным.

27. ДЕШЕВАЯ НЕДОЛГОВЕЧНОСТЬ ВЗАМЕН ДОРОГОЙ ДОЛГОВЕЧНОСТИ.

Заменить дорогой объект набором дешевых объектов, поступившись при этом некоторым качеством (например, долговечностью).

24. ПРИНЦИП "ПОСРЕДНИКА".

а. Использовать промежуточный объект, переносящий или передающий действие.

б. На время присоединить к объекту другой (легкоудаляемый) объект.

2. ПРИНЦИП ВЫНЕСЕНИЯ.

Отделить от объекта мешающую часть (мешающее свойство) или, наоборот, выделить единственно нужную часть или нужное свойство.

В отличие от Приема 1, в котором речь идет о делении объекта на равные части, здесь предлагается делить объект на разные части.

ШАГ 4

4.1.1. Конфликтующие элементы: не менее двух конфликтующих элементов, схема конфликта. Какой элемент можно менять?

Файл в машинных кодах

Место хранения файла

Входные данные

Они взаимодействуют между собой, можно ограниченно менять и то и другое.

Место хранения файла общедоступно, поэтому и файл является общедоступным.

4.1.2. Сформулировать ОВ и ОЗ.

ОВ - момент обращения к программе, время использования программы

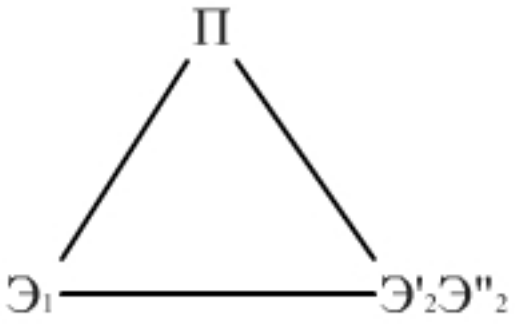
ОЗ - взаимодейтсвие входных данных с программой

4.1.3. Перечень основных ресурсов системы.

Файл в машинных кодах, место хранения, программа, входные данные

4.2.1. Элепольное решение.

Э1 - место хранения, Э2 - файл программы. Э'2 и Э''2 - некоторые части файла программы (по принципу вынесения). Между ними необходимо ввести информационное поле взаимодействия П.



4.2.2. ИКР1: Икс-элемент из ресурсов системы САМ УСТРАНЯЕТ (-), СОХРАНЯЯ (+)

Икс-элемент из ресурсов системы САМ УСТРАНЯЕТ незащищенность, СОХРАНЯЯ доступность.

ШАГ 5

5.1. ПрС. Конфликтующий элемент должен обладать свойством Х, чтобы обеспечить главное требование (+), и должен обладать свойством "АНТИ-Х", чтобы устранить недостаток (-).

Конфликтующий элемент должен обладать свойством входных данных, чтобы оставить программу доступной и обеспечить ее работу, и должен обладать свойством пароля, чтобы защитить программу от несанкционированного доступа.

5.2. ИКР2: Оперативная зона (указать) в течение оперативного времени (указать) должна САМА обеспечивать (указать противоположные макро- или микросостояния).

Взаимодействие входных данных с программой в момент выполнения должно САМО ограничивать несанкционированный доступ к файлу, оставляя его доступным.

ШАГ 6

6.1 Мобилизовать ресурсы. Использовать ресурсы для разрешения конфликта.

В качестве ресурсов можно использовать: размер файла (занимаемое дисковое пространство), количество файлов, тип носителя (места хранения), формат входных данных.

6.2. Использовать информационные фонды: указатели эффектов, ФОП, линии развития в стандартах, законы развития.

пропущен

ШАГ 7

7. Переформулировка (уточнение) задачи.

7.1. если имеется идея (идеи) решения, то необходимо обязательно сформулировать ее (их) недостатки, проверить соответствие полученного решения формулировкам ИКР, приемам и стандартам и далее:

- либо оценить эти недостатки и не точности как не влияющие на работоспособность идеи,

- либо вернуться к рекомендациям по использованию приемов и стандартов,

- либо уточнить ресурсы для поиска решения,

- либо отказаться от предлагаемой идеи и вернуться к шагу1.

Решение 1

По принципу вынесения мы разделим файл на части. По принципу посредника программа не будет работать по отдельности, вынесенные кусок будет играть роль посредника, только с его помощью можно будет пользоваться программой. Вынесенный кусок автор может хранить в удобном для себя месте. Идея заключается в том, что вынесенная часть отдельно переносится автором. Программа не будет работать без нее.

Например, бинарный файл вызывает вынесенную скриптовую часть, без которой работа программы невозможна.

7.2. Если идея кажется удовлетворительной, но не ясно как ее исполнить, то необходимо перейти к формулировке вторичной задачи:

- уточнить целевую (искомую) функцию (объединить функции, перейти к подфункции и др.),

- уточнить или сделать противоположными ограничения,

- перейти к рассмотрению подсистем и сформулировать задачи для них.

7.3. Если найденное решение кажется удовлетворительным и работоспособным, то:

- рассмотреть последствия использования (реализации) предлагаемой идеи,

- рассмотреть задачу с позиции сопутствующих аспектов.

Существенным недостатком предыдущего решения является то, что автору всегда необходимо с собой переносить вынесенную часть программу.

Решение 2

После уточнения ресурсов был выделен новый ресурс - формат исходных данных. Он приводит к еще одному варианту решения задачи. Программа пишется так, чтобы она работала только с определенным форматом входных данных, который знает только автор.

Это решение удовлетворяет формулировке ИКР - Программа САМА в момент обращения к ней защищает себя от доступа всех, кроме ее автора, сохраняя свою доступность. По принципу обратить вред в пользу, мы добавляем вредный фактор (особенный формат входных данных), который дает нам требуемый положительный эффект.

7.4. Остановить анализ, если новая задача не поставлена и нет ее изменений. В противном случае перейти к шагу 1.

Приложение 3. Постановка и решение задачи с использованием алгоритма свертки

Задача о программе вычисления произвольного полинома

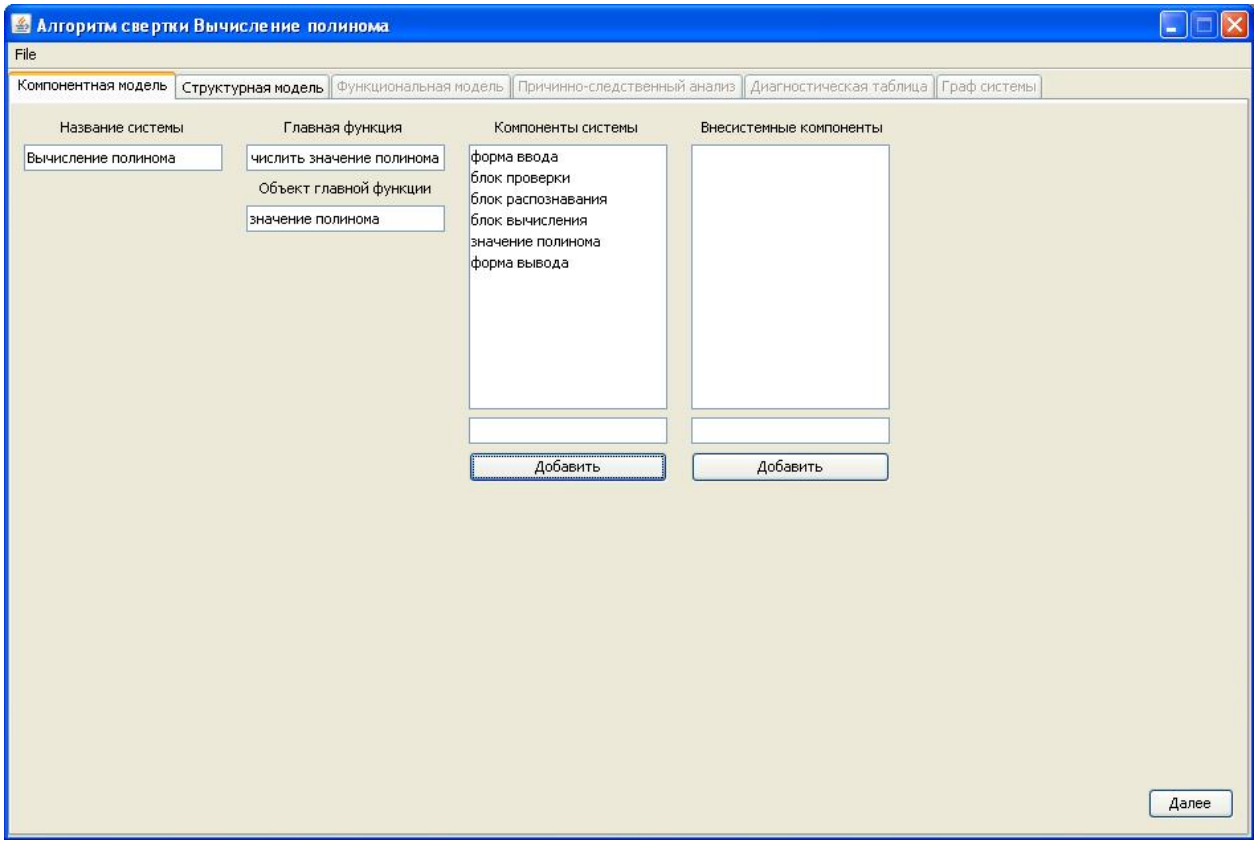
В программе, предназначенной для вычисления значение произвольного полинома, имеется текстовое поле для ввода. Введенная строка затем проверяется на

правильность (действительно ли пользователь ввел полином, а не произвольную строку), а затем распознается (создается модель полинома для его последующего вычисления). Однако, написание такой программы вызывает трудности, возникает большое количество ошибок. Кроме того, усложняется структура выполнения программы. Необходимо упростить программу.

Этап 1. Постановка задачи

Выполним алгоритм свертки системы, чтобы выявить противоречие.

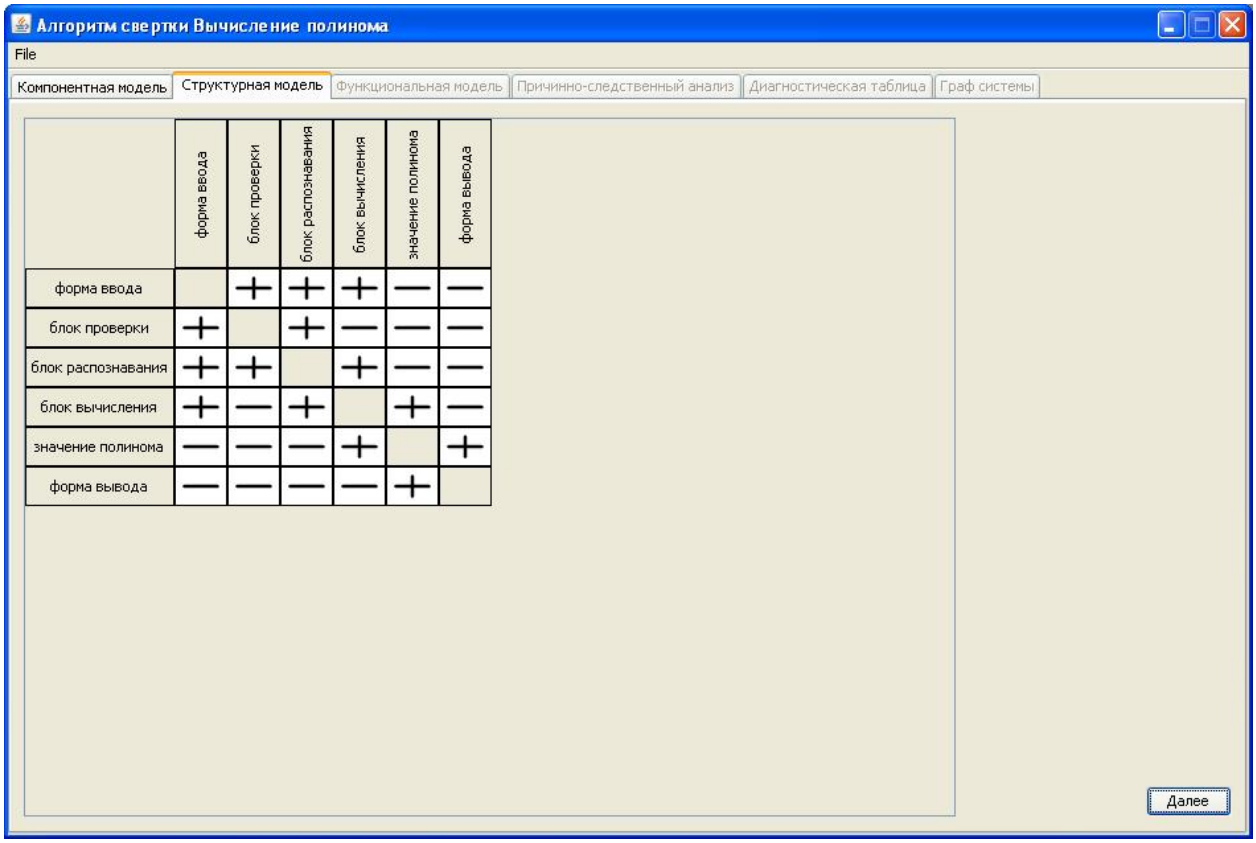
Компонентная модель программы



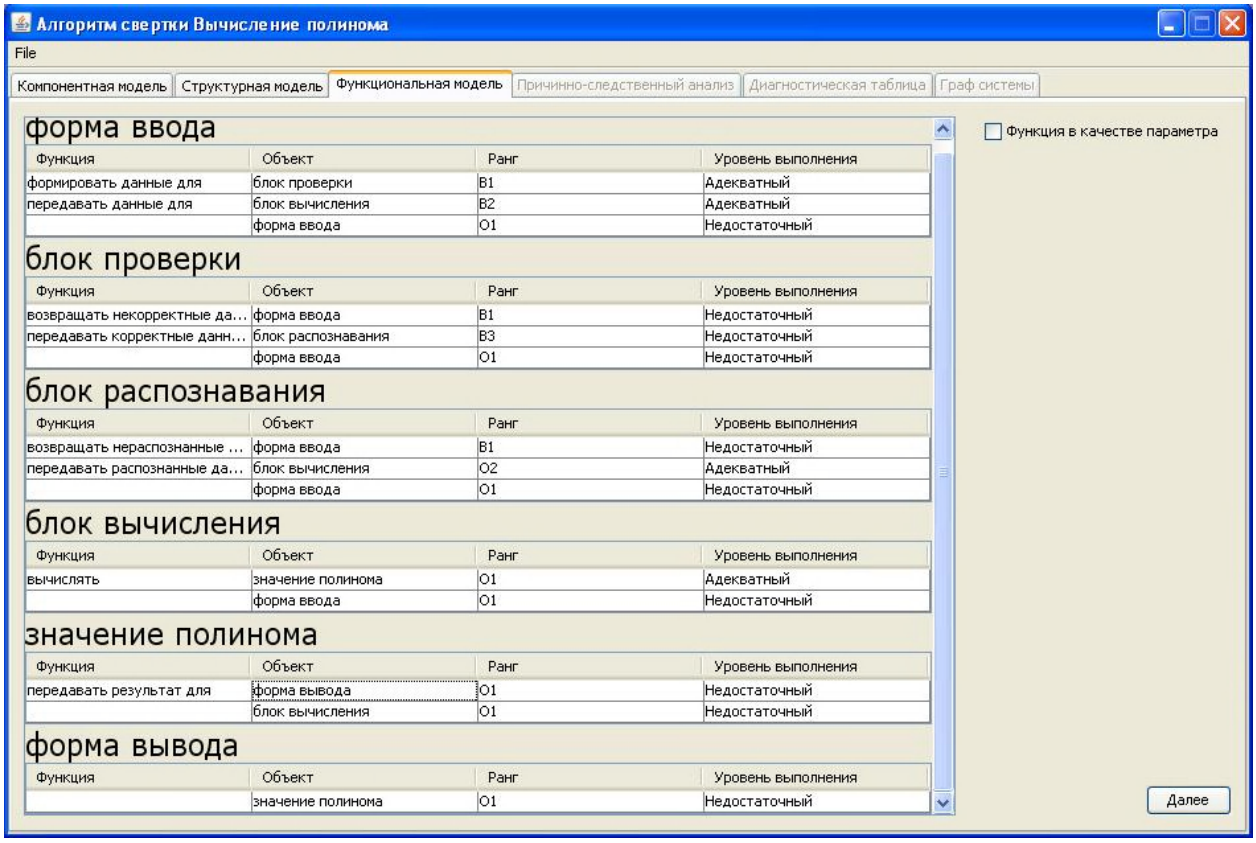
Основными компонентами программы являются: форма ввода, блок проверки, блок распознавания, блок вычисления, значение полинома, форма вывода.

Главной функцией программы является вычисление значения полинома.

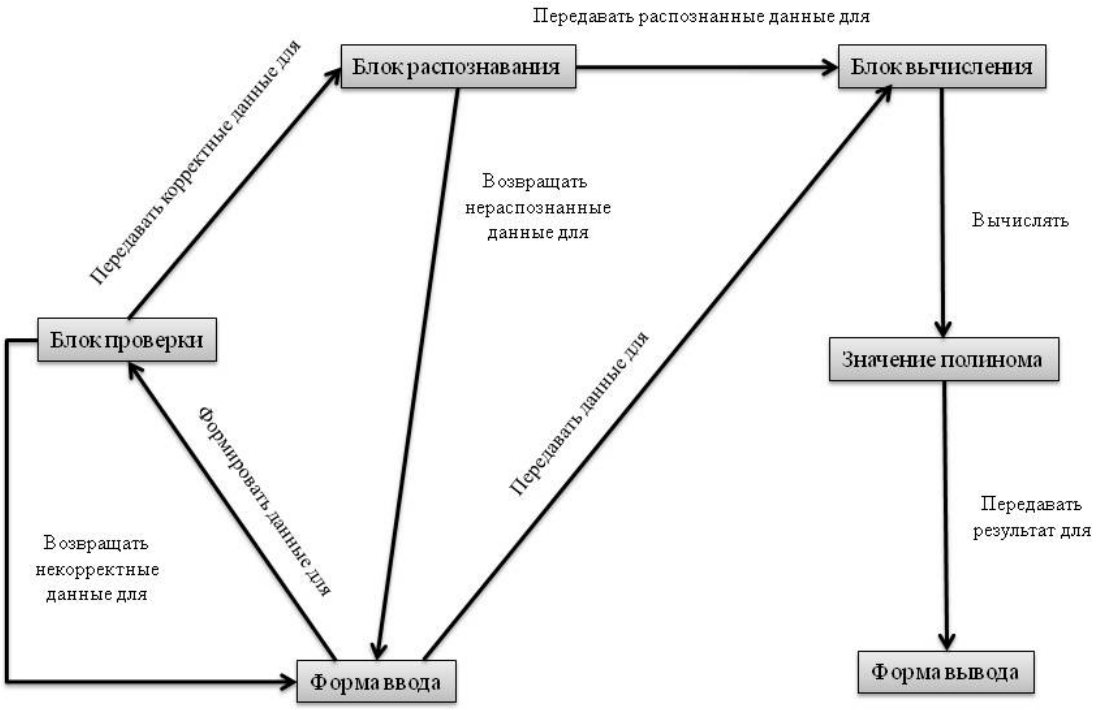
Структурная модель программы



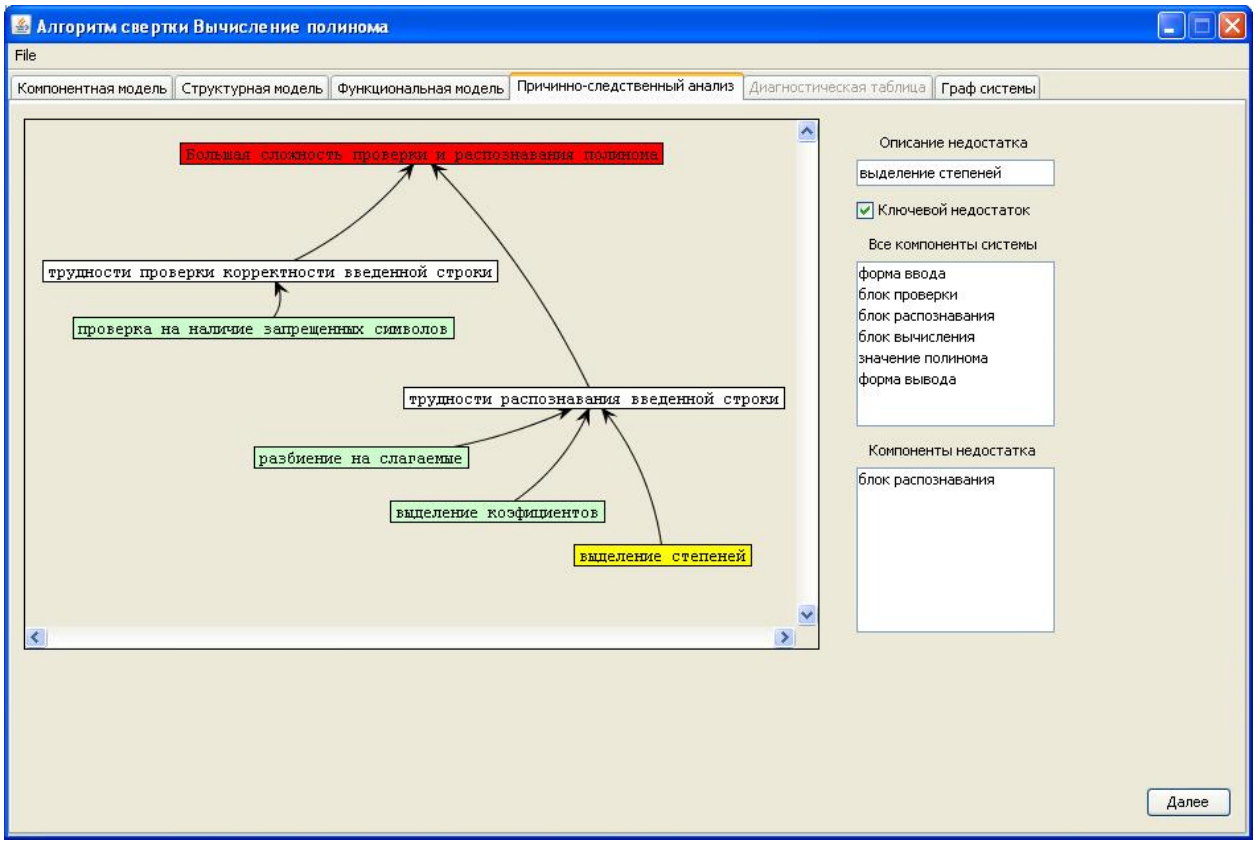
Функциональная модель программы в табличном виде



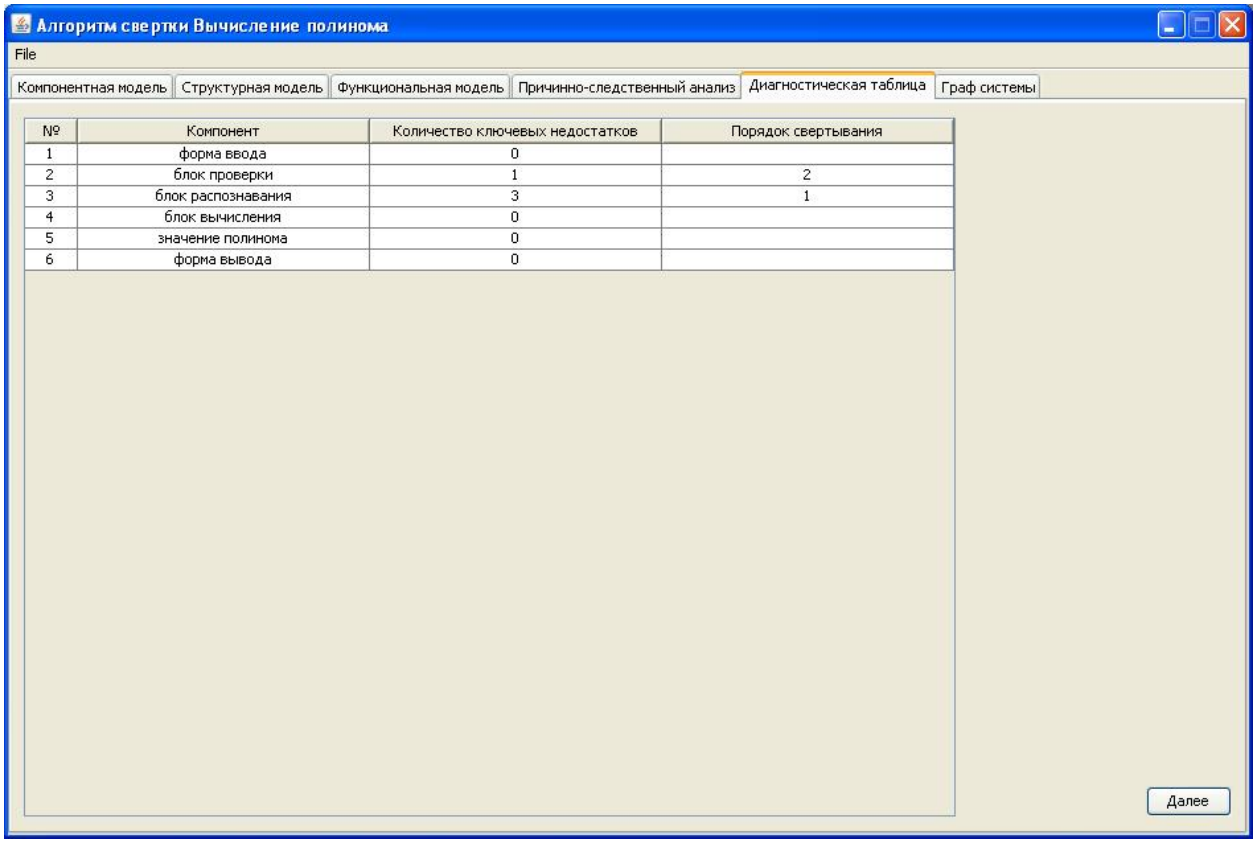
Функциональная модель программы в графическом виде



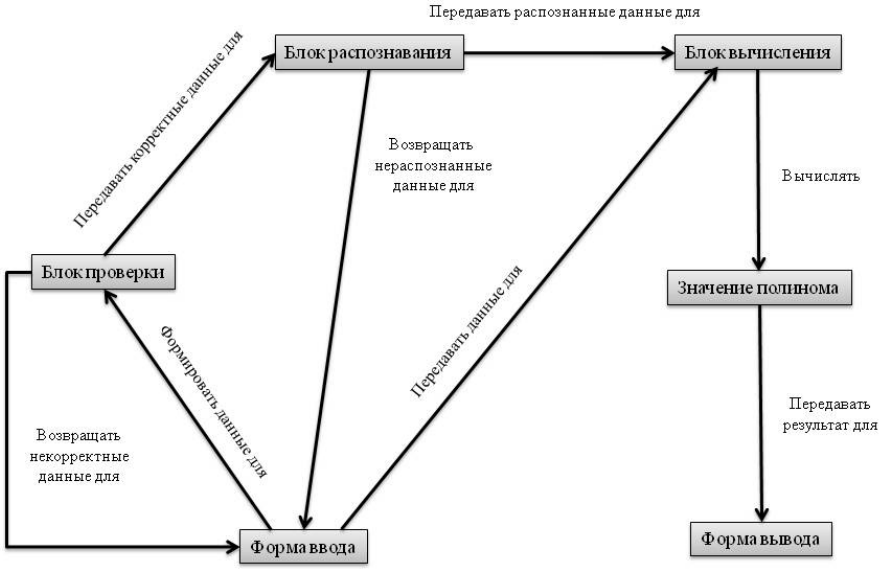
Причинно-следственный анализ



Диагностическая таблица



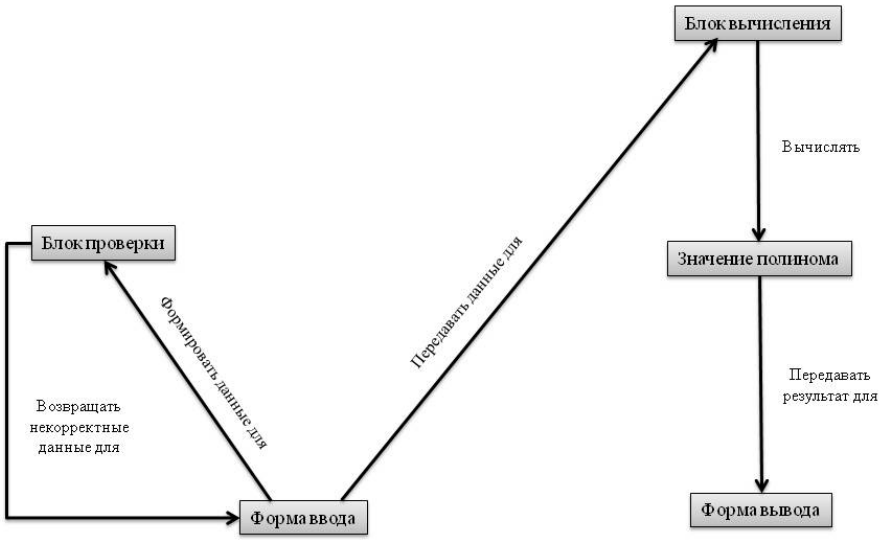
Свертывание элементов программы



Проведем свертывание компонентов согласно порядку, определенному в диагностической таблице.

1. Блок распознавания

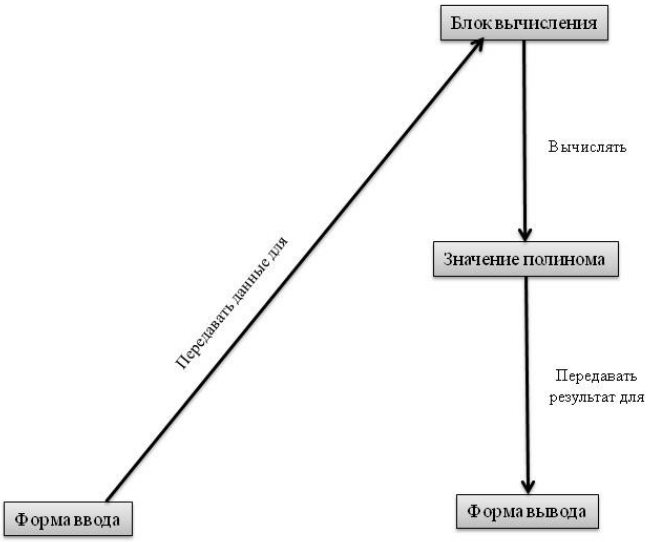
Воспользуемся правилом свертки №3 – элемент можно не делать, если функцию выполняют оставшиеся элементы системы.



Функции, выполняемы оставшимися компонентами системы: передавать распознанные данные для блока вычисления, возвращать нераспознанные данные для формы ввода.

2. Блок проверки

Воспользуемся правилом свертки №3



Функции, выполняемы оставшимися компонентами системы: передавать распознанные данные для блока вычисления, возвращать нераспознанные данные для формы ввода, возвращать некорректные данные для формы вывода.

После завершения алгоритма свертки мы получили функционально-идеальную модель программы.

Сформулируем задачу свертывания:

Вычислять значение полинома с помощью формы ввода и блока вычисления. При этом либо форма ввода, либо блок вычисления должны выполнять следующие функции: распознавать данные для блока вычисления, возвращать нераспознанные или некорректные данные для формы ввода.

Очевидно, эти функции должны выполняться формой ввода.

Задача свертывания в виде противоречия:

Форма ввода не должна накладывать ограничения, чтобы обеспечить возможность задания произвольного полинома, и должна иметь жесткие ограничения, чтобы проверять корректность введенной строки.

Этап 2. Решение задачи по АРИЗ

ШАГ 1

1. Сформулировать главное требование к системе, целевую характеристику.

Необходимо создать такую форму ввода полинома, чтобы она позволяла вводить произвольный полином, и чтобы проверяла введенные данные на корректность и распознавала бы их.

ШАГ 2

2.1. Сформулировать главные функции, которые необходимо выполнить: носитель функции (субъект), действие и/или изменение параметра, объект функции и имеющиеся ограничения.

носитель функции (может быть не известен по условиям задачи): форма ввода

действие и/или изменение параметра (обязательно): задать (создать)

объект функции (обязательно): модель полинома

имеющиеся ограничения: обеспечить произвольность полинома, корректность введенных данных

Ввести полином, обеспечив его произвольность и правильность (корректность) ввода.

2.2.1. Функциональный ИКР. Объект (назвать) САМ делает (описать) в период (назвать) при обязательных условиях (описать).

Модель полинома САМА задается для программы во время ввода пользователем при сохранении произвольности полинома и обеспечении корректности.

2.2.2. ФОП.

пропущен

ШАГ 3

3.1. Противоречие требований (ПрТ): ЕСЛИ ...(указать вносимое изменение).., ТО (+ указать главное требование), НО (−указать нежелательное требование).

ЕСЛИ сделать ввод полинома фиксированным, ТО полином будет введен (будет создана модель полинома), НО не будет обеспечена произвольность ввода.

ЕСЛИ осуществлять ввод полинома в произвольной форме (текстовой строкой), ТО будет обеспечена произвольность ввода полинома, НО проверка ввода и создание модели будут трудны.

3.2.1. Таблица применения типовые приемов разрешения противоречий требований (технических противоречий).

Будем использовать сокращенную таблицу.

Нужно изменить: Удобство изготовления(32)

Ухудшается: Сложность контроля и измерения(37)

По таблице получаем: 32/37 - (6,11,1)

6. ПРИНЦИП УНИВЕРСАЛЬНОСТИ.
Объект выполняет несколько разных функций, благодаря чему отпадает необходимость в других объектах.

11. ПРИНЦИП "ЗАРАНЕЕ ПОДЛОЖЕННОЙ ПОДУШКИ".
Компенсировать относительно невысокую надежность объекта заранее подготовленными аварийными средствами.

1. ПРИНЦИП ДРОБЛЕНИЯ.
а. Разделить объект на независимые части.
б. Выполнить объект разборным.
в. Увеличить степень дробления объекта.

ШАГ 4

4.1.1. Конфликтующие элементы: не менее двух конфликтующих элементов, схема конфликта. Какой элемент можно менять?

Полином

Компоненты формы ввода

Они взаимодействуют между собой, можно менять форму ввода.

4.1.2. Сформулировать ОВ и ОЗ.

ОВ – время ввода полинома

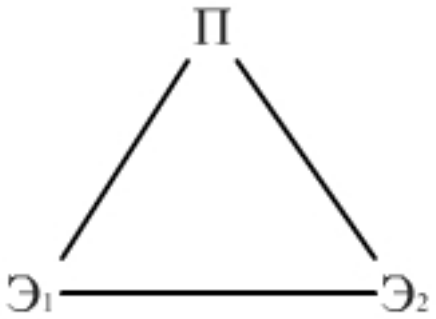
ОЗ – область формы ввода

4.1.3. Перечень основных ресурсов системы.

Полином, форма ввода, модель полинома

4.2.1. Элепольное решение.

Э1 – полином, Э2 – форма ввода. Между ними необходимо ввести поле взаимодействия
П – некоторые правила создания модели полинома.



4.2.2. ИКР1: Икс-элемент из ресурсов системы САМ УСТРАНЯЕТ (-), СОХРАНЯЯ (+)

Икс-элемент из ресурсов системы САМ УСТРАНЯЕТ необходимость проверки на корректность, СОХРАНЯЯ возможность введения произвольного полинома.

ШАГ 5

5.1. ПрС. Конфликтующий элемент должен обладать свойством Х, чтобы обеспечить главное требование (+), и должен обладать свойством "АНТИ-Х", чтобы устранить недостаток (-).

Конфликтующий элемент должен обладать свойство проверки, чтобы обеспечить корректность ввода, и должен обладать свойством произвольности, чтобы гарантировать ввод любого полинома.

5.2. ИКР2: Оперативная зона (указать) в течение оперативного времени (указать) должна САМА обеспечивать (указать противоположные макро- или микросостояния).

Область формы ввода в течение ввода полинома САМА обеспечивает проверку на корректность и гарантирует возможность произвольного полинома.

ШАГ 6

6.1 Мобилизовать ресурсы. Использовать ресурсы для разрешения конфликта.

В качестве ресурсов можно использовать: компоненты формы ввода (например текстовые поля, кнопки), структуру полинома (слагаемые, коэффициенты и степени).

6.2. Использовать информационные фонды: указатели эффектов, ФОП, линии развития в стандартах, законы развития.

пропущен

ШАГ 7

7. Переформулировка (уточнение) задачи.

7.1. если имеется идея (идеи) решения, то необходимо обязательно сформулировать ее (их) недостатки, проверить соответствие полученного решения формулировкам ИКР, приемам и стандартам и далее:

- либо оценить эти недостатки и не точности как не влияющие на работоспособность идеи,

- либо вернуться к рекомендациям по использованию приемов и стандартов,

- либо уточнить ресурсы для поиска решения,

- либо отказаться от предлагаемой идеи и вернуться к шагу1.

Решение

Согласно принципу дробления, структуру полинома следует разделить на части. Естественным образом он разбивается на слагаемые. По принципу заранее подложенной подушки нужно компенсировать ошибки при вводе слагаемых некоторыми средствами формы ввода и модели полинома. На коэффициенты и степени известно ограничение – они могут быть только числами. Это упрощает проверку (раньше при задание всего полинома могли быть введены как числа, так и буквы, обозначающие переменные). Таким образом следует разбить ввод полинома на ввод слагаемых, проверка каждого из которых будет довольно проста.

7.2. Если идея кажется удовлетворительной, но не ясно как ее исполнить, то необходимо перейти к формулировке вторичной задачи:

- уточнить целевую (искомую) функцию (объединить функции, перейти к подфункции и др.),

- уточнить или сделать противоположными ограничения,

- перейти к рассмотрению подсистем и сформулировать задачи для них.

7.3. Если найденное решение кажется удовлетворительным и работоспособным, то:

- рассмотреть последствия использования (реализации) предлагаемой идеи,

- рассмотреть задачу с позиции сопутствующих аспектов.

7.4. Остановить анализ, если новая задача не поставлена и нет ее изменений. В противном случае перейти к шагу 1.