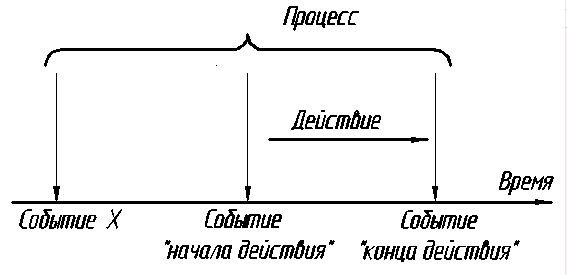
# 1.Предпроектное исследование.

## Основные подходы к построению ИМ.

Системы имитационного моделирования СДС в зависимости от способов представления процессов, происходящих в моделируемом объекте, могут быть дискретными и непрерывными, пошаговыми и событийными, детерминированными и статистическими, стационарными и нестационарными.

Рассмотрим основные моменты этапа создания ИМ. Чтобы описать функционирование СДС надо описать интересующие нас события и действия, после чего создать алфавит, то есть дать каждому из них уникальное имя. Этот алфавит определяется как природой рассматриваемой СДС, так и целями ее анализа. Следовательно, выбор алфавита событий СДС приводит к ее упрощению – не рассматриваются многие ее свойства и действия не представляющие интерес для исследователя.

Событие СДС происходит мгновенно, то есть это некоторое действие с нулевой длительностью. Действие, требующее для своей реализации определенного времени, имеет собственное имя и связано с двумя событиями – начала и окончания. Длительность действия зависит от многих причин, среди которых время его начала, используемые ресурсы СДС, характеристики управления, влияние случайных факторов и т.д. В течение времени протекания действия в СДС могут возникнуть события, приводящие к преждевременному завершению действия. Последовательность действий образует процесс в СДС (Рис. 2.).



***Рис. 1. Взаимосвязь между событиями, действием и процессом.***

В соответствии с этим выделяют три альтернативных методологических подхода к построению ИМ: событийный, подход сканирования активностей и процессно-ориентированный.

## Процесс имитации в РДО.

Для имитации работы модели в РДО реализованы два подхода: событийный и сканирования активностей.

***Событийный подход.***

При событийном подходе исследователь описывает события, которые могут изменять состояние системы, и определяет логические взаимосвязи между ними. Начальное состояние устанавливается путем задания значений переменным модели и параметров генераторам случайных чисел. Имитация происходит путем выбора из списка будущих событий ближайшего по времени и его выполнения. Выполнение события приводит к изменению состояния системы и генерации будущих событий, логически связанных с выполняемым. Эти события заносятся в список будущих событий и упорядочиваются в нем по времени наступления. Например, событие начала обработки детали на станке приводит к появлению в списке будущих событий события окончания обработки детали, которое должно наступить в момент времени равный текущему времени плюс время, требуемое на обработку детали на станке. В событийных системах модельное время фиксируется только в моменты изменения состояний.

Выбор из списка будущих событий ближайшего

Перевод модельного времени вперед

Выполнение программы для данного события

***Рис. 2. Выполнение событий в ИМ.***

***Подход сканирования активностей.***

При использовании подхода сканирования активностей разработчик описывает все действия, в которых принимают участие элементы системы, и задает условия, определяющие начало и завершение действий. После каждого продвижения имитационного времени условия всех возможных действий проверяются и если условие выполняется, то происходит имитация соответствующего действия. Выполнение действия приводит к изменению состояния системы и возможности выполнения новых действий. Например, для начала действия обработка детали на станке необходимо наличие свободной детали и наличие свободного станка. Если хотя бы одно из этих условий не выполнено, действие не начинается.

Вход

Выбрать из спиcка очередное действие

Выполнить действие, обновив состояние системы

Условие выполняется?

Все действия просмотрены?

Печать итогового отчета

Завершен ли прогон?

Да

Нет

Нет

Нет

Да

***Рис. 3. Блок-схема реализации***

***подхода сканирования активностей.***

## Основные положения языка РДО.

В основе системы РДО – «Ресурсы, Действия, Операции» – лежат следующие положения:

* Все элементы сложной дискретной системы (СДС) представлены как ресурсы, описываемые некоторыми параметрами.
* Состояние ресурса определяется вектором значений всех его параметров; состояние СДС – значением всех параметров всех ресурсов.
* Процесс, протекающий в СДС, описывается как последовательность целенаправленных действий и нерегулярных событий, изменяющих определенным образом состояния ресурсов; действия ограничены во времени двумя событиями: событиями начала и конца.
* Нерегулярные события описывают изменение состояния СДС, непредсказуемые в рамках продукционной модели системы (влияние внешних по отношению к СДС факторов либо факторов, внутренних по отношению к ресурсам СДС). Моменты наступления нерегулярных событий случайны.
* Действия описываются операциями, которые представляют собой модифицированные продукционные правила, учитывающие временные связи. Операция описывает предусловия, которым должно удовлетворять состояние участвующих в операции ресурсов, и правила изменения ресурсов в начале и конце соответствующего действия.
* При выполнении работ, связанных с созданием и использованием ИМ в среде РДО, пользователь оперирует следующими основными понятиями:
* **Модель** - совокупность объектов РДО-языка, описывающих какой-то реальный объект, собираемые в процессе имитации показатели, кадры анимации и графические элементы, используемые при анимации, результаты трассировки.
* **Прогон** - это единая неделимая точка имитационного эксперимента. Он характеризуется совокупностью объектов, представляющих собой исходные данные и результаты, полученные при запуске имитатора с этими исходными данными.

**Проект** - один или более прогонов, объединенных какой-либо общей целью. Например, это может быть совокупность прогонов, которые направлены на исследование одного конкретного объекта или выполнение одного контракта на имитационные исследования по одному или нескольким объектам.

**Объект** - совокупность информации, предназначенной для определенных целей и имеющая смысл для имитационной программы. Состав объектов обусловлен РДО-методом, определяющим парадигму представления СДС на языке РДО.

Объектами исходных данных являются:

* типы ресурсов (с расширением .rtp);
* ресурсы (с расширением .rss);
* образцы операций (с расширением .pat);
* операции (с расширением .opr);
* точки принятия решений (с расширением .dpt);
* константы, функции и последовательности (с расширением .fun);
* кадры анимации (с расширением .frm);
* требуемая статистика (с расширением .pmd);
* прогон (с расширением .smr).

Объекты, создаваемые РДО-имитатором при выполнении прогона:

* результаты (с расширением .pmv);
* трассировка (с расширением .trc).

## 1.4. Постановка задачи.

Поддержка полноценного событийного подхода в РДО была реализована сравнительно недавно, и в системе остались некоторые функции, требующие доработки.

В частности это касается инструкции планирования событий. С помощью функции Planning() мы можем в явном виде управлять планированием событий в системе, но не можем передавать планируемому событию никаких дополнительных параметров, возможно требующихся для адекватного построения модели (к примеру, это может быть в явном виде заданное количество деталей, одновременно поступивших на обработку, или коэффициент, учитывающий сложность обработки). К тому же при описании паттерна события мы можем явно указывать для него список параметров. К слову, поддержка данного механизма реализована в описании активностей, но у пользователей РДО должна появиться возможность передавать в качестве параметров не только численные константы, но и арифметические выражения (чего сейчас нет у активностей)

Это представляет интерес для пользователей, так как событийный подход является самым гибким способом описания моделируемой системы.

В качестве примера модели, который система РДО должна поддерживать в явном виде имеет смысл выбрать простейшую систему гибкого производственного участка ввиду ее наглядности и универсальности. Сразу отметим, что в данной работе не ставилось цели промоделировать процесс обслуживания, поэтому простейшая модель служит лишь иллюстрацией и тестом функциональности системы.

Для полноценного использования нового функционала системы РДО соответствующих изменений требует и справочный материал, встроенный в систему. Т.е. пользователи РДО должны иметь возможность прочитать во встроенной справке о всех нововведениях и найти там соответствующие примеры.

# РАЗРАБОТКА ТЕХНИЧЕСКОГО ЗАДАНИЯ НА СИСТЕМУ

## Основания для разработки

* Задание на курсовой проект.

## Назначение разработки

Основная цель данного курсового проекта – реализовать передачу параметров событиям в системе имитационного моделирования РДО ( т.е. разработать синтаксис, и логику обработки этих параметров внутри RAO Studio.

## Характеристики объекта автоматизации.

РДО – язык имитационного моделирования, включающий на данный момент подход сканирования активностей, процессный и событийный подход.

## Требования к программе или программному изделию

При написании образца активности или события пользователь может в разделе $Parameters указывать необходимые для моделирования параметры образца. Поскольку образцы представляют собой структуру Базы знаний системы и не используются сами по себе, при заведении от них активностей необходимо явно указывать численные значения, передаваемые в качестве параметров. События также являются элементом Базы знаний, поэтому необходимо реализовать, чтобы при планировании события пользователь мог указывать инструкцию, позволяющую передавать параметры событиям.

**$Pattern** Событие\_начала\_обслуживания **: event**

**$Parameters** Число\_одновременно\_поступивших\_заявок **: integer** = 1

**$Relevant\_resources**

\_Станок: Станок **Keep**

**$Body**

\_Станок

**Convert\_event**

Событие\_окончания\_обслуживания**.planning**(**time\_now** + Длительность\_обслуживания( 30, 50 ), Количество\_шпинделей\_станка );

количество\_в\_очереди += Число\_одновременно\_поступивших\_заявок;

**$End**

**$Pattern** Событие\_окончания\_обслуживания : **event trace**

**$Parameters** Обрабатываются\_одновременно : **integer** = 1

**$Relevant\_resources**

\_Станок: Станок **Keep**

**$Body**

\_Станок

**Convert\_event**

количество\_обслуженных += Обрабатываются\_одновременно;

**if** (Станок.количество\_в\_очереди > Обрабатываются\_одновременно)

{

количество\_в\_очереди -= Обрабатываются\_одновременно;

Событие\_начала\_обслуживания**.planning**(**Time\_now** + Интервал\_прихода(30),Среднее\_число\_одновременно\_поступающих\_заявок(6,1) );

}

**else**

{

состояние\_станка = Свободен;

}

**$End**

### Требования к надежности

Основное требование к надежности направлено на поддержание в исправном и работоспособном ЭВМ на которой происходит использование программного комплекса RAO-Studio.

### Условия эксплуатации

* Аппаратные средства должны эксплуатироваться в помещениях с выделенной розеточной электросетью 220В ±10%, 50 Гц с защитным заземлением.

### Требования к составу и параметрам технических средств

Программный продукт должен работать на компьютерах со следующими характеристиками:

* объем ОЗУ не менее 256 Мб;
* объем жесткого диска не менее 20 Гб;
* микропроцессор с тактовой частотой не менее 400 МГц;
* монитор не менее 15” с разрешением от 800\*600 и выше;

### Требования к информационной и программной совместимости

Данная система должна работать под управлением операционных систем Windows 2000, Windows XP, Windows Vista и Windows 7.

### Требования к маркировке и упаковке

Не предъявляются.

### Требования к транспортированию и хранению

Не предъявляются.

## Требования к программной документации

Необходимо включить описание инструкции передачи параметров в «Справку по языку РДО» .

## Стадии и этапы разработки

* Предпроектное исследование.
* Концептуальный этап проектирования.
* Технический этап проектирования.
* Рабочий этап проектирования.

## Порядок контроля и приемки

Контроль и приемка поддержки передачи параметров должны осуществляться в процессе проверки функциональности (апробирования) системы имитационного моделирования на тестовом примере модели в соответствии с требованиями к функциональным характеристикам системы.

# Концептуальный этап проектирования.

## Диаграмма компонентов.

Система имитационного моделирования РДО безусловно является сложной и статически, и динамически. На это указывает сложная иерархическая структура системы со множеством различных связей между компонентами и ее сложное поведение во времени.

Ярко выраженная иерархическая структура и модульность системы определяют направление изучения системы сверху вниз. Т.е. мне необходимо применять принцип декомпозиции нужных модулей до тех пор, пока не будет достигнут уровень абстракции, представление на котором нужных объектов не нуждается в дальнейшей детализации для решения данной задачи.

Для отображения зависимости между компонентами системы РДО и выделения среди них модернизируемых служит соответствующая диаграмма в нотации UML.

***Рис. 4. Упрощенная диаграмма компонентов.***

Базовый функционал представленных на диаграмме компонентов:

rdo\_kernel реализует ядровые функции системы. Не изменяется при разработке системы.

RAO-studio.exe реализует графический интерфейс пользователя. Не изменяется при разработки системы.

rdo\_repository реализует управление потоками данных внутри системы и отвечает за хранение и получение информации о модели. Не изменяется при разработке системы.

rdo\_mbuilder реализует функционал, используемый для программного управления типами ресурсов и ресурсами модели. Не изменяется при разработке системы.

rdo\_simulator управляет процессом моделирования на всех его этапах. Он осуществляет координацию и управление компонентами rdo\_runtime и rdo\_parser. Не изменяется при разработке системы.

rdo\_parser производит лексический и синтаксический разбор исходных текстов модели, написанной на языке РДО. Модернизируется при разработке системы.

rdo\_runtime отвечает за непосредственное выполнение модели, управление базой данных и базой знаний. Модернизируется при разработке системы.

В дальнейшем компонент rdo\_parser описывается более детально.

# ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

## Доработка синтаксиса инструкции планирования событий.

Для создания лексического анализатора в системе РДО используется генератор лексических анализаторов общего назначения Bison, который преобразует описание контекстно-свободной LALR(1) грамматики в программу на языке C для разбора этой грамматики. Для того, чтобы Bison мог разобрать программу на каком-то языке, этот язык должен быть описан контекстно-свободной грамматикой. Это означает, необходимо определить одну или более синтаксических групп и задать правила их сборки из составных частей. Например, в языке C одна из групп называется `выражение'. Правило для составления выражения может выглядеть так: "Выражение может состоять из знака `минус' и другого выражения". Другое правило: "Выражением может быть целое число". Правила часто бывают рекурсивными, но должно быть по крайней мере одно правило, выводящее из рекурсии.

Наиболее распространённой формальной системой для представления таких правил в удобном для человека виде является форма Бэкуса-Наура (БНФ, Backus-Naur Form, BNF), которая была разработана для описания языка Algol 60. Любая грамматика, выраженная в форме Бэкуса-Наура является контекстно-свободной грамматикой. Bison принимает на вход, в сущности, особый вид БНФ, адаптированный для машинной обработки.

В правилах формальной грамматики языка каждый вид синтаксических единиц или групп называется символом. Те из них, которые формируются группировкой меньших конструкций в соответствии с правилами грамматики, называются нетерминальными символами, а те, что не могут разбиты -- терминальными символами или типами лексем. Мы называем часть входного текста, соответствующую одному терминальному символу лексемой, а соответствующую нетерминальному символу -- группой.

Каждая группа, так же как и её нетерминальный символ, может иметь семантическое значение. В компиляторе языка программирования выражение обычно имеет семантическое значение в виде дерева, описывающего смысл выражения.

Доработка синтаксиса для инструкций планирования событий заключается в добавлении арифметического выражения, соответствующего передаваемым параметрам:

**<имя\_события>.Planning(<арифметическое\_выражение>) (<арифметическое\_выражение>1, <арифметическое\_выражение>2 …<арифметическое\_выражение>n)**

Синтаксическая диаграмма данной группы представлена на листе 4 курсового проекта:



***Рис. 5.Синтаксическая диаграмма инструкции «Планирование события».***

**имя события** – это имя одного из событий, описанных в модели, которое должно быть запланировано;

**арифметическое выражение** – обычное арифметическое выражение, в состав которого могут входить параметры релевантных образцу и глобальных ресурсов.

В синтаксис арифметических выражений также было добавлено значение по-умолчанию:



***Рис. 5.Синтаксическая диаграмма инструкции «Арифметическое выражение».***

При разборе входного текста необходимо добавить ряд действий, которые выполнятся при распознавании текста. К примеру, при разборе инструкции планирования, при разборе параметров нужно 1-е выражение в списке интерпретировать как время следующего события, а остальные – как значения передаваемых параметров:

ArithmContainer::Container::const\_iterator arithmIt = pArithmList->getContainer().begin();

if (arithmIt == pArithmList->getContainer().end())

{

PARSER->error().error(@1, rdo::format(\_T("Не указано время планирования события: %s"), eventName.c\_str()));

}

LPRDOFUNArithm pTimeArithm = \*arithmIt;

ASSERT(pTimeArithm);

++arithmIt;

LPArithmContainer pParamList = rdo::Factory<ArithmContainer>::create();

ASSERT(pParamList);

while (arithmIt != pArithmList->getContainer().end())

{

pParamList->addItem(\*arithmIt);

++arithmIt;

}

pEvent->setParamList(pParamList);

## Разработка алгоритма передачи параметров.

Существующий в системе алгоритм планирования событий является базой для добавления новых функций. На данный момент в алгоритме предусмотрено позднее связывание, благодаря которому мы можем обращаться к объектам rdoRuntime (для поиска запланированных событий) В свою очередь в компоненте rdoParse необходимо добавить функцию, осуществляющую передачу параметров на этапе позднего связывания объектам Runtim-a.

Следует добавить, что целью работы также было не просто добавление статических параметров, но и осуществление возможности передавать вместе с инструкцией планирования не только константы, но и арифметические выражения. Для реализации этой возможности в пространстве имен rdoParse был добавлен новый класс, который является по сути унифицированным арифметическим выражением(более высокий уровень абстракции) и наследование от него.

 ***Рис. 6.Алгоритм передачи параметров событиям.***

При отсутствии

# РАБОЧЕЕ ПРОЕКТИРОВАНИЕ

## Изменения в файлах синтаксического анализатора

Для реализации в среде имитационного моделирования нового инструмента разработанного на концептуальном и техническом этапах проектирования, в первую очередь необходимо добавить новые нетерминальные символы в грамматический анализатор.

### Синтаксический анализ объявления конструкции «параметры события»

Инструкция планирования события в системе РДО имеет вид:

planning\_statement

: ИДЕНТИФИКАТОР '.' Инструкция планирования'(' время планирования ')'';'

Для того, чтобы передать в планируемое событие параметры, необходимо разработать синтаксическую конструкцию. Так как в проекте также стояла задача передать в событие не просто константу, а обеспечить возможность передавать функции (например, параметры, заданные через закон распределения), то было принято решение передавать в инструкцию планирования такое выражение:

planning\_statement

: ИДЕНТИФИКАТОР '.' Инструкция планирования'(' лист арифметических выражений ')'';'

Для реализации данной инструкции в файл синтаксического анализатора был добавлен новый нетерминальный символ, описывающий «лист арифметических выражений»:

arithm\_list

: /\* empty \*/

{

LPArithmContainer pArithmContainer = rdo::Factory<ArithmContainer>::create();

ASSERT(pArithmContainer);

$$ = PARSER->stack().push(pArithmContainer);

}

| arithm\_list\_body

;

arithm\_list\_body

: fun\_arithm

{

LPArithmContainer pArithmContainer = rdo::Factory<ArithmContainer>::create();

LPRDOFUNArithm pArithm = PARSER->stack().pop<RDOFUNArithm>($1);

ASSERT (pArithmContainer);

ASSERT (pArithm);

pArithmContainer->setSrcText(pArithm->src\_text());

pArithmContainer->addItem (pArithm);

$$ = PARSER->stack().push(pArithmContainer);

}

| arithm\_list\_body ',' fun\_arithm

{

LPArithmContainer pArithmContainer = PARSER->stack().pop<ArithmContainer>($1);

LPRDOFUNArithm pArithm = PARSER->stack().pop<RDOFUNArithm>($3);

ASSERT (pArithmContainer);

ASSERT (pArithm);

pArithmContainer->setSrcText(pArithmContainer->src\_text() + \_T(", ") + pArithm->src\_text());

pArithmContainer->addItem (pArithm);

$$ = PARSER->stack().push(pArithmContainer);

}

| arithm\_list\_body ',' error

{

PARSER->error().error(@3, \_T("Ошибка в арифметическом выражении"));

}

;

## 4.2.Изменения в пространстве имен rdoParse.

Добавлен новый класс, описывающий контейнер арифметических выражений:

//! Список арифметических выражений

//! \details Используется для передачи параметров при вызове событий и функций

OBJECT(ArithmContainer) IS INSTANCE\_OF(RDOParserSrcInfo)

{

DECLARE\_FACTORY(ArithmContainer);

public:

typedef std::vector<LPRDOFUNArithm> Container;

CREF(Container) getContainer() const { return m\_arithmList; }

void addItem (CREF(LPRDOFUNArithm) pArithm);

private:

ArithmContainer();

virtual ~ArithmContainer();

Container m\_arithmList;

};

В механизм позднего связывания была добавлена функция, обеспечивающая связь с объектами rdoRuntime и позволяющая реализовать передачу параметров объектам Runtime-а:

STL\_FOR\_ALL\_CONST(pEvent->getParamList()->getContainer(), paramIT)

{

LPRDOFUNArithm pParam = \*paramIT;

if (m\_currParam < pPattern->m\_paramList.size())

{

rdoRuntime::LPRDOCalc pSetParamCalc;

LPRDOParam pPatternParam = pPattern->m\_paramList[m\_currParam];

ASSERT(pPatternParam);

if (pParam->typeInfo()->src\_info().src\_text() == \_T("\*"))

{

if (!pPatternParam->getDefault()->defined())

{

RDOParser::s\_parser()->error().push\_only(pPatternParam->src\_info(), rdo::format(\_T("Нет значения по-умолчанию для параметра '%s'"), pPatternParam->src\_text().c\_str()));

RDOParser::s\_parser()->error().push\_only(pPatternParam->src\_info(), rdo::format(\_T("См. параметр '%s', тип '%s'"), pPatternParam->src\_text().c\_str(), pPatternParam->getTypeInfo()->src\_info().src\_text().c\_str()));

RDOParser::s\_parser()->error().push\_done();

}

rdoRuntime::RDOValue val = pPatternParam->getDefault()->value();

ASSERT(val);

pSetParamCalc = rdo::Factory<rdoRuntime::RDOSetPatternParamCalc>::create(

m\_currParam,

rdo::Factory<rdoRuntime::RDOCalcConst>::create(val)

);

}

else

{

LPTypeInfo pTypeInfo = pPatternParam->getTypeInfo();

ASSERT(pTypeInfo);

rdoRuntime::LPRDOCalc pParamValueCalc = pParam->createCalc(pTypeInfo);

ASSERT(pParamValueCalc);

pSetParamCalc = rdo::Factory<rdoRuntime::RDOSetPatternParamCalc>::create(

m\_currParam,

pParamValueCalc

);

}

ASSERT(pSetParamCalc);

pActivity->addParamCalc(pSetParamCalc);

++m\_currParam;

}

else

{

RDOParser::s\_parser()->error().push\_only(pParam->src\_info(), rdo::format(\_T("Слишком много параметров для события '%s' при планировании события '%s'"), pEvent->name().c\_str(), pEvent->name().c\_str()));

RDOParser::s\_parser()->error().push\_done();

}

}

}

# Заключение

В рамках данного курсового проекта были получены следующие результаты:

* + 1. Проведено предпроектное исследование системы имитационного моделирования РДО.
    2. На этапе концептуального проектирования системы с помощью диаграммы компонентов нотации UML укрупненно показано внутреннее устройство РДО и выделены те компоненты, которые потребуют внесения изменений в ходе этой работы.
    3. На этапе технического проектирования доработан синтаксис событий, который представлен на синтаксической диаграмме. С помощью диаграммы классов разработана архитектура новой системы. С помощью блок-схемы разработан алгоритм, реализующий в системе РДО механизм передачи параметров событиям.
    4. На этапе рабочего проектирования написан программный код для реализации спроектированных раннее алгоритмов работы и архитектуры компонента rdo\_parser системы РДО. Проведены отладка и тестирование нового функционала системы, в ходе которых исправлялись найденные ошибки.
    5. Для демонстрации новых возможностей системы модель, представленная на этапе постановки задачи, была реализована на в системе РДО. Результаты проведения имитационного исследования позволяют сделать вывод об адекватной работе новой функции системы.
    6. Все внесенные в систему изменения справочной информации по системе РДО, что позволяет пользователям оперативно получать справку по новым функциям системы.

Поставленная цель работы достигнута в полном объеме.

# Список использованных источников

* + 1. RAO-Studio – Руководство пользователя, 2007 [[http://rdo.rk9.bmstu.ru/forum/viewtopic.php?t=900](http://rdo.rk9.bmstu.ru/forum/viewforum.php?t=900)].
    2. Справка по языку РДО (в составе программы) [<http://rdo.rk9.bmstu.ru/forum/viewforum.php?f=15>].
    3. Емельянов В.В., Ясиновский С.И. Имитационное моделирование систем: Учеб. пособие. – М.: Изд-во МГТУ им.Н.Э. Баумана, 2009. – 584 с.: ил. (Информатика в техническом университете).
    4. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению. ГОСТ 19.201-78.
    5. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. ГОСТ 19.701-90. Условные обозначения и правила выполнения.
    6. Бьерн Страуструп. Язык моделирования C++. Специальное издание. Пер. с англ. – М.: ООО «Бином-пресс», 2007 г. – 1104 с.: ил.