С.Ю. Бутузов, А.В. Крючков
(Академия ГПС МЧС России; e-mail: butuzov_s_yu@mail.ru)

СЕРВИС УДАЛЁННОЙ РАЗРАБОТКИ СПЕЦИАЛЬНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ В ИНТЕРЕСАХ МЧС РОССИИ

Предлагается создание сервиса удалённой разработки программного обеспечения автоматизированных систем техносферной безопасности.
Ключевые слова: программное обеспечение, удалённая разработка.

S.Yu. Butuzov, A.V. Krjuchkov

REMOTE SERVICE SPECIFIC SOFTWARE DEVELOPMENT FOR THE BENEFIT OF EMERCOM OF RUSSIA

Creation of remote service specific software development for automated systems of technosphere safety are proposed.
Key words: software, remote development.

Статья поступила в редакцию Интернет-журнала 23 сентября 2013 г.

Технологии техносферной безопасности в современных условиях требуют адекватного реагирования на возникновения ***чрезвычайных ситуаций (ЧС)***. Это означает, что сотрудники, выполняющие работы по их мониторингу и ликвидации, нуждаются в современных средствах поддержки их деятельности как в ходе подготовки к ликвидации ЧС, так и в ходе оперативного реагирования на них. Для устойчивого функционирования таких средств поддержки необходимо применение современных ***информационных технологий (ИТ)***, обеспечивающих как необходимую скорость получения оперативной и иной информации специалистами, так и обеспечение их всевозможными видами расчётов и аналитических данных.

Основные функции современных ИТ, реализующие конкретные требования сотрудников МЧС, обеспечиваются специфическим ***программным обеспечением (ПО)***, составными частями которого являются отдельные приложения. Их функции и состав обеспечиваются требованиями к выполнению функциональных задач конкретных сотрудников МЧС в ходе их работы.

В развитии современных ИТ в последнее время (примерно с 2009 г.) наметился новый тренд, получивший название ***облачных вычислений***. Его применение обосновано, прежде всего, резким снижением затрат как на инфраструктуру ИТ, так и на персонал, обслуживающий данную инфраструктуру. Для его практической реализации используются модели отношений заказчика и поставщика, основанные на сервисах (или услугах). Заказчик заказывает необходимые ему мощности или лицензии, а поставщик предоставляет заказчику требуемые ресурсы, самостоятельно выстраивая отношения с поставщиками оборудования и ПО.

Вместе с тем в вопросах разработки специфического или ***специального ПО (СПО)***, в соответствии с ГОСТ 34.003-90 [1], данный тренд не получил должного развития, несмотря на появление таких услуг, как PaaS (Platform as a Service) и APaaS (APlication a Service). Поэтому актуальным остаётся вопрос о создании ***сервиса удалённой разработки*** в интересах сотрудников МЧС, осуществляющих деятельность в интересах обеспечения техносферной безопасности.

Предполагаемый к разработке сервис может быть охарактеризован термином ***"программирование как сервис"***. Он основан на 15-летнем практическом и статистическом исследовании процесса программирования в различных по численности коллективах программистов (от 5 до 1500 человек), и предназначен для повышения производительности труда непосредственных "исполнителей" кода, технических писателей, менеджеров низшего и среднего звена. В разработанном подходе широко применяется принцип ***"повторного использования кода"***. Сутью данного сервиса является перенесение в облако коллективов программистов, работающих у перспективных клиентов в качестве сотрудников компаний. Наибольшая эффективность данного процесса (в экономии средств и ресурсов заказчиков и разработчиков) может достигаться в случае миграции в облако крупных систем, требующих переработки при изменении базовой платформы разработки для приложений.

В его основе лежит предположение о том, что ПО крупных ***автоматизированных систем управления производством (АСУП)*** может быть представлено в виде набора однотипных по составу и структуре компонент. Их сейчас принято называть приложениями, ***автоматизированными рабочими местами (АРМ)***, специальным программным обеспечением АРМ. Но, по сути, речь идёт о той их программной части, которая непосредственно используется пользователем и пишется программистом для пользователя (иногда вопреки требованиям ТЗ). Предлагаемый набор методов представляет собой реализуемую на любом инструментальном средстве модель повторно используемого кода, полученную на основе статистического и практического анализа работающего кода и методов его создания программистами.

Структура компонентов АСУП или задач (СПО конкретных АРМ, или, в целях отсутствия путаницы, ***единичных программных систем – ЕПС***) в рамках предложенного подхода состоит из трёх групп параметров. То есть ЕПС рассматривается с трёх основных позиций: структура данных приложения, структура интерфейса пользователя (та её часть, которая непосредственно взаимодействует с ним в ходе его работы) и структура исходного кода ЕПС ([2]). Такой подход, подкреплённый статистическим анализом практических реализаций в коде различными программистами, позволил разработать набор методов и подходов к созданию модели повторно используемой части кода для ЕПС в АСУП.

Новые требования к АРМ новых руководителей заставляют дорабатывать готовые отлаженные программы "на лету" по нескольку раз в год. Иногда необходимо полностью перерабатывать их структуру и текст. Если полностью сле-

довать букве и духу [3] или иных подобных документов (в том числе отраслевых, которые запрещали вести разработку без утверждённого ТЗ), то эту работу вообще невозможно сделать в сроки, которые требуются пользователям. На одни переговоры заказчиков и разработчиков и подписание соответствующих документов уходят недели, а иногда и месяцы.

Процесс написания программ, с точки зрения программиста, может быть представлен так. Стараясь свести к минимуму свои усилия, программист станет разрабатывать инструментарий (библиотеку подпрограмм) на том средстве, которое потребует от него наибольшего количества сдаваемых к сроку приложений (или в терминах ряда документов, основанных на отменённых ГОСТах 24 серии, задач). При увеличении нагрузки для избранного им инструментального средства программист усовершенствует инструментарий для более глубокой автоматизации своего труда. Иными словами, кирпичи в здании приложения станут больше (иногда превратятся в плиты) и будут состоять из более качественной глины. Для 2-го (3-го, 5-го) средства программист попробует создать нечто подобное.

Предположим, в проекте у нас 5 программистов на одном инструментальном средстве, которые вечно спорят на SCRUM-сессиях о том, как надо писать ту или иную программу. Двое договорились, а остальные – нет. В результате даже в таком незначительном коллективе появиться 4 подхода к повторно используемому коду на одном инструментальном средстве. Через полгода после сдачи появились новые требования, и проект надо дорабатывать. А программисты ушли. Если они были сотрудниками отдела, ответственного за разработку программ, в крупной компании, то в самом оптимистичном случае приходящие им на смену люди доработают один из их инструментариев.

Теперь рассмотрим более крупное подразделение с несколькими филиалами в разных регионах страны. Проблема с исправлением и доработкой кода остаётся. В случае роста системы, написанной на едином инструментальном средстве, проблема обостряется, так как растёт число инструментариев, сопровождать которые уже сложно даже большим коллективом программистов.

Тоже самое можно сказать и о разработке систем с нуля (крупных проектах). В крупных системах, не стремящихся к самоорганизации, где количество ЕПС может стремиться к нескольким сотням, количество методов и подходов к реализации требований пользователей сильно возрастает. А процесс написания кода не может быть ни тарифицирован, ни проконтролирован менеджментом проекта. Поэтому возникла ***проблема "крупных проектов"***.

Решение проблемы может быть найдено в случае построения модели инструментариев, построенной на основе статистического анализа и опыта практического проектирования и реализации ЕПС, создаваемые прикладным программистом, имеют очень разное назначение. Поэтому предлагаемые в статье методы разработки СПО относятся и к ряду специфических, например, научных программ, разрабатываемых в единичных экземплярах, или программ, которые требуют работы с хорошо проработанным набором структур данных, мало изменяемых на протяжении всего срока эксплуатации данных ЕПС. В целях более

точного определения диапазона приложений по применимости созданной методологии авторами был введён термин ***"базового класса задач автоматизации" (БКЗА)***. Ввод термина, как и в случае с ЕПС, вынужденный, так как нормативные документы (например, ГОСТ 19781-90, п. 3 [4]) недостаточно полно определяют требуемые понятия. БКЗА – это достаточно представительный класс ЕПС, в котором характер эксплуатации ЕПС пользователем предполагает наличие в структуре данных одной главной таблицы с данными. Таблица, как правило, содержит учётные карточки, главный журнал или т.п. Вокруг данного учётного материала строится костяк приложения и его основные функции, которые доступны пользователю через интерфейс.

По оценкам авторов, крупные ИС содержат от 70 до 90 % таких ЕПС. В некоторых случаях, отдельные приложения могут быть приведены к подобным структурам данных без ущерба для их дальнейшей эксплуатации пользователями. Наличие БКЗА существенно облегчает построение дальнейших предположений о сути методов. Это связано с тем, что, во-первых, все приложения в крупной системе могут рассматриваться с одинаковых позиций. Поэтому модель повторно используемого кода, разработанная для одной ЕПС, годится и для остальных. А во-вторых, все ЕПС при этом, независимо от реальных структур данных в предметной области, могут иметь одинаковую структуру данных, разработанную заранее (до старта процесса разработки). Это позволяет создать набор одинаковых по логике работы программ (инструментарий), использование которого пользователями в различных приложениях, а также различными программистами при разработке новых ЕПС с помощью тех же самых или новых инструментальных средств будет одинаковым.

Прежде, чем описывать детали, целесообразно упомянуть о генераторах кода и современных средствах разработки (Google Apps и т.п. [5]). Зачем делать инструментарий и методологию создания ПО, когда всё есть? Как правило, разработчики средств разработки предоставляют разработчикам приложений достаточно универсальный инструмент, который не ориентирован на вполне конкретный класс задач автоматизации (объектов разработки). Более того, с точки зрения продавцов этих продуктов, их потребление будет только расти в случае, когда в них инструментарий разработчика (прикладного программиста) будет представлен более разнообразными методами реализации отдельных компонент приложения в исходном коде. Кроме того, средства генерации программ требуют наличия специфического для конкретного инструментального средства входного языка, на котором записываются задания на генерацию. Два прикладных программиста создадут для похожих по структуре данных приложений два разных задания на генерацию. При этом интерфейс пользователя будет также различен, даже если его неоднократно обсуждать в SCRUM-сессиях.

Вот отрывок из статьи Брайана Кернигана "Почему я не люблю программировать на Паскале", написанной ещё в 1981 году: "Каждый делает то, во что горазд, а в результате возникает проблема переносимости программ, написанных на собственных "доморощенных" Паскалях." [6].

Помимо этого, хочется сказать, что статьи о повторно используемом коде носят, к сожалению, более описательно статистический, нежели технический характер. В них много данных о том, что повторно использовать код можно и это могло бы принести пользу, но нет анализа того, как писать повторно используемый код пусть даже на вполне конкретном средстве разработки (например, на Perl или Java), чтобы им потом было удобно пользоваться тем, кто пожелает. Нет и анализа структуры приложений и оценок применимости различных подходов к тому, как писать для разных по структуре данных и составу приложений подобный код. Видимо, именно поэтому Керниган выдумал Clipper для персоналок (который в 5-й версии имел всего 360 зарезервированных слов), хотя до этого вполне обходился Си. А Стив Макконнелл пишет книгу за книгой о том, что просто необходимо превратить программирование из творческого искусства в инженерно индустриальный процесс. Этот переход возможен только в случае широкого использования "одинаковых деталей", которые используются всеми участниками процесса. Сейчас трудно представить себе строителя, делающего ремонт в квартире, который самостоятельно изготавливает саморезы, уголки, составляет рецептуру клея и собирает для этого материал где-то на пустыре. А с программированием всё обстоит именно так.

Первая часть методов относится к разработке и поддержанию структуры данных предметной области приложения, относящегося к БКЗА и метаданными приложения. Данные предметной области в подавляющем большинстве случаев, по своей сути, представляют собой набор таблиц БД для хранения информации приложения (ЕПС). Тем не менее, фактически этот набор таблиц можно свести к 5 таблицам. 2 из них будут содержать информацию о структурном дереве конкретной ЕПС (групповой уровень, дерево информационной схемы приложения ДИСП), а 3 – моделировать карточку учёта.

Такая структура данных позволяет создавать одинаковый по составу и структуре интерфейс программиста (набор *API*-вызовов или библиотеку подпрограмм) на любом средстве до начала процесса разработки конкретного приложения. В результате для приложений из БКЗА от проектирования БД в традиционно принятой форме в перспективе вообще можно будет отказаться. А описание данных предметной области будет выполняться аналогично заполнению БД информацией.

Обычно работа по проектированию структур данных предметной области – это около 15 % всего времени разработки. При этом наличие даже несложной по составу и функциям программы (мастера), реализующей набор функций для работы с такой структурой данных, позволяет сократить это время в 5 раз.

Методы разработки СПО в этой части представляют собой описание алгоритмов разработки программ работы с деревом информации приложения и карточкой ***информационной единицы хранения (ИЕХ)***. Это та часть кода, которая представляет собой компонент ввода и корректировки данных и компонент работы с метаинформацией приложения. Генераторы кода данную часть методов разработки всегда перекладывают на разработчика приложений.

Помимо этого в эту группу попадают группы модулей для работы с метаданными приложения, то есть таблицами БД, содержащими информацию о меню (интерфейсом пользователя), расчётами и отчётами, фильтрами и индексами. Все они хранятся вместе или отдельно от приложения в зависимости от конкретной реализации технологии "клиент-сервер", "толстый клиент" и т.п. Концепция паспорта информационной единицы хранения предметной области позволяет сделать работу с данными различных предметных областей одинаковой. Подобная организация данных позволяет достаточно точно (погрешность до 10 *Кб*) определять объём необходимой квоты приложения, а при наличии убедительной статистики и прогнозировать её. При этом статистика использования одного приложения позволяет масштабировать её результаты, то есть использовать эти данные для оценки квот АСУП, состоящих из сотен ЕПС.

Вторая группа методов относится к интерфейсу пользователя. В эту группу включены метод реализации интерфейса группового уровня или ***дерева информационной схемы приложения (ДИСП)***, а также метод реализации семантических элементов (функций) пользовательского интерфейса ЕПС, на этапе её работы с карточкой ИЕХ.

Никакая обобщенная официальная классификация специального ПО (по ГОСТ 34.003-90) не составлена. Не составлена она и для инструментальных средств его реализации в соответствии с показателем, который бы определял трудозатраты одного программиста или групп программистов на весь процесс программирования от получения ими утверждённого ТЗ до утверждения кода системы руководителем команды тестировщиков.

Для описания сути состава повторно используемого кода введём следующую ***классификацию***. Разделим существующие ***средства программирования*** условно на ***процедурные*** и ***непроцедурные***. Эти понятия новые.

К ***непроцедурным*** отнесем те средства, которые позволяют программировать при помощи больших готовых блоков или использовать их внутренний интерфейс (меню, диалог, команды) в качестве основы для создания готового продукта. Примерами могут служить Excel, Access, PDM-системы, САПР и их аналоги и даже некоторые текстовые редакторы (Word и др.), позволяющие "писать программы" при помощи автоматически записываемых этими средствами макросов.

К ***процедурным*** средствам отнесем те из них, которые заставляют программиста описывать детали СПО. Например, создавать табличные редакторы в Clipper 5.01, Visual FoxPro, а также работать с формами или таблицами БД в Delphi, PHP, C#, Visual C++, Visual J, JavaScript и т.п.

Такая классификация является условной и соответствует спектру действий программиста внутри инструментального средства по реализации СПО. Если говорить о крайних точках такой классификации в сторону "увеличения" возможностей программиста по разработке (количества выполненных им исходных текстов) своих систем до уровня ОС, то мы получим машинно-ориентированный язык из нулей и единиц. А если в сторону расширения возможностей пользователя, то получим абстрактную систему без программирова-

ния (СБП-систему), которая могла бы стать предельным случаем автоматизации программирования.

Под СБП-системой здесь и далее будет пониматься некий гипотетический программный продукт, носящий на себе черты многих инструментальных средств, к которым можно отнести ряд функций CASE-систем, PDM-систем и других, подобных им по реализации возможностей программиста. Исходя из опыта практической работы с подобными средствами при реализации ЕПС, можно сделать ряд выводов об общих структурных частях и составу как ЕПС, так и систем программирования.

Если следовать терминам системного анализа и теории образов, то для определения состава СБП-системы необходимо провести декомпозицию объекта (образа) ЕПС. В результате выяснится, что СБП-система, предназначенная для реализации кода ПО в ЕПС, может состоять из следующих частей:

- компонент (модуль) описания экранов;
- компонент (модуль) ввода и корректировки данных;
- компонент (модуль) описания расчётов;
- компонент (модуль) описания отчетов;
- компонент (модуль) описания условий (ввода, расчётов и вывода);
- компонент (модуль) описания фильтров и индексов;
- компонент (модуль) описания меню.

Кроме этого в её состав могут входить:

- компонент (модуль) описания и работы с архивами;
- компонент (модуль) "быстрого" ввода данных;
- диалоговые средства для работы в сети;
- средства (компоненты), обеспечивающие удобство пользователя при работе с данными на различных уровнях (копирование файлов, калькулятор, работа с дисками и т.п.).

Разумеется, такое разделение исходного кода ЕПС условно, но оно позволяет говорить о реализации повторно используемых компонент в ЕПС для различных предметных областей языком процентов

Помимо этого, привязка к таким компонентам ЕПС позволяет гораздо более точно оценить загрузку и нагрузку работников как в случае одного, так и в случае многих программистов. Написание компонент и их отладка происходят независимо от процесса разработки ЕПС. Сборка ЕПС из данных компонент, выполненная неоднократно, позволяет сократить число невынужденных ошибок в сборках (читай – в ЕПС). Повторное и многократное использование компонент позволяет:

- сократить практически до 0 число ошибок;
- увеличить значение в отношении – число ЕПС / один программист – на порядок;
- сократить влияние факторов персонала на разработку кода при его дальнейшем "обслуживании" (совершенствовании, доработке и модификации);
- обеспечить в перспективе стандартизацию прикладного ПО или СПО (во всяком случае той его части, которая попадает в БКЗА).

Собственно говоря, данная группа методов представляет собой детальные описания реализации перечисленных компонент на различных средствах программирования (один компонент и одно средство программирования – один метод). В качестве платформы для реализации могут использоваться любые средства программирования любых компаний: PHP, Java, C#, Jelastic, Google Apps Engine, Microsoft Azure, IBM Jazz, IBM Rational/ClearQuest, Eclipse и т.п.

Написание подобной "обвязки" ДИСП и реализация интерфейса пользователя вместе с отладкой в коде составляет до 70 % от всего времени, которое программист тратит на написание кода приложения. Предварительная реализация компонент приложения (в виде концепции СБП-системы) позволит сократить время реализации требований пользователя в приложении до 10 раз при разработке с нуля, и – от 50 до 100 % при модификации.

Создание крупных ИС, в случае использования данного подхода, может дать ещё более значительный выигрыш во времени, трудозатратах и финансах. Кроме того, сложная система управления, создаваемая ранее для реализации подобных проектов, становится ненужной.

Состав работ по созданию сервиса определяется составом компонент СБП-системы. Их число умножается на число средств разработки, которое будет охвачено сервисом.

На этапе создания прототипов компонент на первом средстве разработки потребуются значительные усилия нескольких человек. Один компонент – один разработчик. По времени примерно 3-5 месяцев, хотя по опыту, скорее всего 2 последних месяца уйдут на доводку. Здесь можно (и желательно) использовать готовое решение вендора, суть доработки которого заключается в создании ряда мастеров (диалоговых помощников) для диалогового изменения кода, написании редактора расчётов и соединении данных, полученных из ТЗ с мастерами создания кода.

Получение прототипа СБП-системы на конкретном средстве разработки позволит составить детальное ТЗ для каждого из компонент с целью их реализации на других средствах разработки. Наличие ЧТЗ на компоненты позволит оценить стоимость и целесообразность развития основанного на методологии сервиса в том или ином направлении. Вендоры могли бы и самостоятельно заняться подобной реализацией (и инвестированием в неё) при соблюдении ряда условий.

Состав сервиса таков. Дополнительные инструменты взаимодействия с потенциальными "покупателями" сервиса. "Покупатели" – это, прежде всего, подразделения МЧС, которые имеют в своём составе штатные отделы разработки программ, отделы и дирекции проектов, управления автоматизации или развития ИТ, в которых работают штатные программисты, от услуг которых нельзя отказаться. Хотя это было бы очень желательно.

Для реализации сервиса могут потребоваться следующие компоненты:

- модуль управления требованиями;
- модуль утверждения требований;
- модуль взаимодействия с клиентами (аналог call-центра + простая CRM-система для выписывания счетов и учёта поставок);
- модуль управления программистами (статистика работы);
- генератор данных, переводящий требования ТЗ клиента во входную информацию генератора приложений;
- генератор кода приложения, основанный на методологии;
- БД готовых решений;
- БД предметных областей.

Модуль управления требованиями необходим для получения ТЗ от пользователя. В настоящий момент в прототипе сервиса реализовано сокращённое ТЗ по ГОСТ 15.201-2003. В приложениях к нему должны вводиться и выводиться частные ТЗ на автоматизированные рабочие места (АРМ). Проект договора на работы по системе также формируется в данном модуле. Более детальные описания модуля есть в текстовом виде, но в силу ограниченности места здесь не приводятся. Предполагается в процессе развития сервиса дополнять выход данного модуля отдельными частями, связанными с реализацией структуры того или иного вида технических документов (ГОСТ, DITA и т.п.).

Модуль утверждения требований необходим для согласования с заказчиком документов и отдельных требований. Он может быть и составной частью модуля взаимодействия с клиентами, однако вынесен в отдельную единицу, так как предполагает наличие данных по "движению" разработанных в модуле документов. Его выход – это вход для генератора данных. Оба модуля работают с клиентами в браузере.

Модуль взаимодействия с клиентами необходим для учёта заявок на работы, интереса перспективных покупателей, поставок и "отгрузки" конкретным клиентам конкретных заказов. Здесь же осуществляется маркетинг и консультирование, а также учёт и статистический анализ по этим направлениям. Он является вспомогательным, а не основным, и независим от остальных модулей, так как хотя и получает, но не выдаёт данных ни в один из перечисленных здесь модулей. Данные в него попадают через операторов.

Модуль управления программистами – своеобразный менеджер проектов. Он отслеживает ход выполнения конкретных заказов конкретными программистами. Частично такая функциональность уже неоднократно реализовывалась, но для полноценной автоматизации "фабрики программирования" необходимо видимо будет в перспективе создать вполне определённый код. Существующие системы основаны на занесении данных программистами в систему. Отслеживание их активности в определённых местах позволило бы собирать информацию и выстраивать статистику в более автоматическом режиме. Он является вспомогательным, а не основным, и независим от остальных модулей, так как хотя и получает, но не выдаёт данных ни в один из перечисленных здесь модулей.

Генератор данных переводит требования ТЗ клиента во входную информацию генератора приложений. Его входом является выход модуль утверждения требований. Выходом является БД предметных областей. Генератор преобразует данные, полученные от пользователей в браузере в паспорта ИЕХ, записывая их в БД с учётом данных о заказчике.

Генератор кода приложения представляет собой улучшенный генератор кода, собирающий ЕПС из компонент, реализованных в конкретном PaaS-средстве с учётом требований конкретного заказчика. Входом модуля является БД предметных областей и БД готовых решений, а выходом – конкретный код конкретного проекта конкретного заказчика. На стадии запуска сервиса в стартапе это может быть решение вендора, привязанное к данным сервиса.

Кроме того, сервису понадобится архив проектов, который по идее необходим для полноценного функционирования сервиса в продвинутом варианте. А для работы с архивом проектов нужен специальный модуль.

БД готовых решений содержит информацию о компонентах СБП-систем, реализованных на различных конкретных PaaS-средствах.

БД предметных областей содержит информацию о паспортах ИЕХ различных предметных областей, по которым велась разработка.

Сервис удалённой разработки не требует наличия программистов в подразделениях МЧС при разработке СПО. Следовательно, влияние персонала на его разработку снижается. Оценки применения данного сервиса по методике оценки COCOMO II 2000 года (модели оценки трудоёмкости разработки программ) [7] показали, что только за счёт уменьшения коэффициентов влияния по линии персонала трудоёмкость разработки СПО снижается в 10 раз.

Таким образом, создание сервиса удалённой разработки приложений в интересах МЧС значительно снизит зависимость подразделений МЧС от некачественного СПО, факторов риска, связанных с разрабатывающим СПО персоналом, а также будет способствовать стандартизации данных и кода приложений, используемых сотрудниками. Это, в свою очередь, приведёт к экономии средств и снижению рисков в сфере информационного обеспечения техносферной безопасности.

Литература

1. ***ГОСТ*** 34.003-90. Информационная технология. Комплекс стандартов на автоматизированные системы. Термины и определения.

2. ***Крючков А.В.*** Математическая модель процесса создания СПО крупной ИС в терминах системного анализа // Научно-технический сборник ОАО Концерн "Системпром". № 1 (3). 2013.

3. ***ГОСТ Р*** ИСО/МЭК 12207-99. Процессы жизненного цикла программных средств.

4. ***ГОСТ*** 19781-90. Обеспечение систем обработки информации программное. Термины и определения.

5. http://jeck.ru/labs/deep/lec_6.html.

6. ***Эрик С. Реймонд.*** Новый словарь хакера. М.: ЦентрКом, 1996.

7. ***Вендров А.М.*** Проектирование программного обеспечения экономических информационных систем. М.: Финансы и статистика, 2005.