А.В. Крючков
(Академия ГПС МЧС России; e-mail: hook66@list.ru)

ОБОБЩЕНИЕ ОПЫТА СИНТЕЗА СПЕЦИАЛЬНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА РАЗЛИЧНЫХ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВАХ

Обобщён опыт работы программистов по синтезу специального программного обеспечения автоматизированных систем управления предприятиями, включая системы безопасности. Даётся качественное описание компонент системы без программирования на основе обобщений, связанных с информационной единицей хранения и универсальным перечнем семантических элементов интерфейса.

Ключевые слова: программное обеспечение, автоматизированные системы управления предприятиями.

A.V. Kruchkov

SUMMARIZING THE EXPERIENCE OF SYNTHESIS SPECIAL SOFTWARE FOR DIFFERENT PROGRAMMING LANGUAGES

Generalized experience of programmers on the synthesis special software of automated enterprise management systems, including security systems. Qualitative description of the system components without programming, based on generalizations associated with the information storage unit and a universal list of semantic interface elements is given.

Key words: software, automated enterprise control systems.

Статья поступила в редакцию Интернет-журнала 31 марта 2015 г.

Синтез ***специального программного обеспечения (СПО) автоматизированных систем управления предприятиями (АСУП***) требует наличия определённых ***информационных технологий (ИТ),*** в состав которых включены ***системы программирования (СП)*** и ***языки программирования (ЯП***). Учитывая, что СПО для крупной АСУП состоит из СПО для отдельных автоматизированных ***рабочих мест (АРМ),*** будем называть СПО, созданное для одного из АРМ, ***единичной программной системой (ЕПС).***

Первым этапом синтеза ЕПС всегда является анализ требований тех пользователей, для которых АРМ предназначено. Опыт показывает, что иерархические требования к системе ЕПС, образующих СПО крупной АСУП, можно объединить в единый свод данных и правил их формирования. Такой объект можно назвать ***"паспортом информационной единицы хранения (ИЕХ)"*** определённой предметной области. Учитывая, что АРМ работает с ***базой данных (БД)*** или таблицей БД, для синтеза СПО необходимо использовать абстрактную или реальную ***систему управления БД (СУБД***).

Для работы с ЕПС на конкретном рабочем месте приложение должно выполнять ряд функций. Анализ различных проектов и их реализаций позволил выявить ряд сходных черт в семантике интерфейсной части АРМ в СПО крупных АСУП. В связи с этим целесообразно обобщить эти данные в виде определённого перечня или набора правил работы с ИЕХ на АРМ конкретного поль-

зователя, с тем, чтобы затем в процессе синтеза новых программных систем в рамках СПО АСУП или совершенствования их работы можно было бы заранее разрабатывать инструментарий приложений вне зависимости от его предметной области.

Семантическим элементом интерфейса (СЭИ) уместно называть то действие ПО, которое пользователь выбирает, общаясь с ПО, и которое реализует одну понятную ему функцию. Такое действие обычно совершается при нажатии кнопки, переходу по гиперссылке или при выборе пункта меню, хотя возможны и иные способы предоставления пользователю функций ввода команд приложению в рамках конкретной ЕПС.

На основании анализа данных по разработанному СПО, информацию о котором автору удалось получить лично в виде исходного кода ЕПС, был сформирован ***универсальный перечень СЭИ (УПСЭИ)*** для абстрактной ЕПС, сгруппированный по категориям. Перечень позволяет не только разрабатывать новые приложения, но и достаточно точно оценивать качество сдаваемого программного продукта заказчиками, не имеющими достаточной квалификации в области синтеза СПО. В результате применения УПСЭИ в процессе синтеза СПО возможно решение задачи преемственности в разработке. Но для корректного синтеза СПО, помимо УПСЭИ, необходимо введение понятия ***"системы без программирования" (СБП-системы)***, основанной, в том числе, и на функциях УПСЭИ.

Рассмотрим более подробно процесс синтеза СПО АСУП, с точки зрения работ, которые необходимо выполнять коллективу-разработчику. При этом анализ и обобщение опыта синтеза СПО на различных инструментальных средствах (ЯП, СП, СУБД, ***языках высокого уровня – ЯВУ***, web-технологиях) будем проводить с помощью терминологии процедурно-ориентированной и объектно-ориентированной методологий разработки ПО.

Общие задачи разработчиков по проведению работ в рамках синтеза СПО АСУП описаны в стандартах, касающихся жизненного цикла ПО [1, 2]. Тем не менее, при использовании процедурно-ориентированной методологии программирования сам процесс синтеза ЕПС по личному опыту автора выглядел примерно так:

1. Проведение опроса потенциальных заказчиков на предмет возможного синтеза СПО.
2. Составление плана взаимодействия между организациями разработчика и заказчиками.
3. Получение исполнителем разработчика задания на разработку.
4. Предварительное изучение предметной области в соответствии с заданием и уточнение деталей.
5. Составление разрешения на обработку данных автоматизированным образом.
6. Определение возможных трудозатрат на написание кода, алгоритмизацию, создание документации, системы помощи пользователю при его работе с программой. Корректировка плана взаимодействия и составление плана реализации данного проекта конкретными исполнителями.

7. Написание ***технического задания (ТЗ)***, его согласование и утверждение заказчиком. Детали ТЗ описывают внешнюю модель данных, порядок её отображения, выходные данные проекта и их представление в виде отчетов, дополнительные функции, реализуемые в проекте, и характер их реализации.

8. Написание постановки задачи на основе ТЗ.

9. Согласование и утверждение постановки задачи внутри организации-разработчика. Представление согласованной постановки задачи заказчику на утверждение.

10. Предварительный и детальный структурный анализ предметной области, определение стиля создаваемого для СПО интерфейса пользователя и предварительная алгоритмизация – определение структурных частей СПО, порядка их взаимодействия и составление алгоритмов верхнего уровня. Составление личного плана реализации СПО.

11. Написание программного кода в соответствии со структурными компонентами, определенными на предыдущем этапе. Следование личному плану (или его нарушение).

12. Отладка программного кода.

13. Создание документации.

14. Согласование, регистрация и утверждение документации внутри организации заказчика.

15. Проведение совместных испытаний СПО на макетном участке. Проверка заказчиком соответствия функциональности представленного кода требованиям ТЗ. Подписание соответствующих формальных документов о доработке СПО или завершении его разработки.

16. Устранение недостатков в программном коде СПО.

17. Установка СПО на персональной ПЭВМ пользователя.

18. Отправка технической документации заказчику.

19. Проведение обслуживания и совершенствования СПО.

Первые десять пунктов соответствуют этапу анализа требований в том виде, в котором его принято описывать в современной технической литературе. Дальнейшие этапы создания программного продукта в данном списке процессов имеют вполне просматриваемое соответствие с [2]. Помимо этого следует заметить, что при проведении работ с использованием ***объектно-ориентированного программирования (ООП)*** изменение претерпят 10 и 11 пункты данного списка. С одной стороны, процесс алгоритмизации в объектно-ориентированном процессе не предусматривается. А с другой – процесс написания кода может быть заменен описанием спецификаций СПО. Фактически для детализации процесса синтеза СПО программистом необходимо подробно изучить пункты 11 и 12 настоящего списка.

Начало и середина 90-х годов 20-го века не один раз изменили многое в процессах создания исходного кода ЕПС. Появились технологии создания ПО, коллективы программистов, методики оценки их работы, стандарты по уровням доверия организациям-разработчикам. Программист как работник или физическое лицо, исполняющее обязанности по должности "программист",

получая право на творчество, несет ответственность за создаваемый им код. С этим обстоятельством связано то, что многие объединения программистов часто носят случайный характер, и каждый из них предпочитает работать в одиночку. Более того, в технической литературе пишут о том, что специалист-одиночка может при определенных условиях сделать для реализации конкретного проекта гораздо больше коллектива программистов [3], на чем основывается и подход в кадровой политике компании-разработчика. Используемые при этом средства программирования такой человек определяет сам, руководствуясь при этом рекламой коллег и знакомых. Свой труд он протоколирует частично, причем часто специально этого не делает для защиты авторских прав. Разумеется, в процессе эксплуатации написанное им ПО будет эксплуатироваться недостаточно эффективно или вообще не будет эксплуатироваться, что в случае получения программистом денег за проект его совершенно не интересует. Недостаточно эффективно при этом означает, что без него обычно никто с этим ПО без его разработчика не справится.

Для создания программ специалисты-одиночки используют различные инструментальные средства программирования. Никакая обобщенная официальная классификация для СПО и инструментальных средств его реализации не составлена. Существующие в настоящее время инструментальные средства программирования в недостаточной степени удовлетворяют потребности программиста-разработчика в стандартных функциях (реализованных в качестве структурных элементов СП или СУБД). Перечни инструментальных средств, реализующих определённый подход к программированию, носят "эксклюзивный" характер. Зачастую составители их классификаций не желают признавать правомочность своих коллег в их составлении. Это вызвано следующими причинами. С одной стороны, различия в подходах, реализующих базовую парадигму программирования, между инструментальными средствами, с точки зрения использующего их программиста, достаточно условны, особенно когда дело доходит до реализации конкретного ***программного проекта ЕПС (ПП ЕПС)***. С другой стороны – инструментальные средства, чтобы лучше продаваться, реализуют в себе несколько подходов. Кроме того, реализованные различными компаниями версии инструментальных средств имеют многочисленные технические особенности. Они исключают совместное использование исходных кодов ПП ЕПС одинаковой предметной области, реализованные на одинаковых, с точки зрения парадигм, ЯВУ.

Разделим существующие средства программирования условно на процедурные и непроцедурные. Заранее заметим, что к процедурным языкам программирования данные понятия не имеют отношения. 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, системы автоматизированного проектирования 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 детально описывать процессы, которые выше названы интерфейсом. Например, создание табличного редактора в Clipper 5.01, Visual FoxPro и т.п. или команд работы с БД в Delphi, FrontPage, Visual C, Visual J (рис. 1).

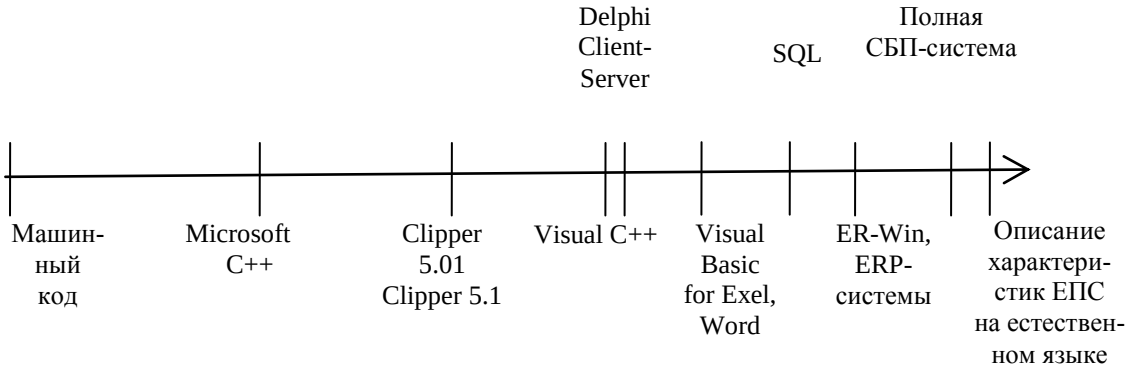

Рис. 1. Условное распределение систем программирования по степени приближения к реальному миру используемых ими ЯВУ

Данный уровень классифицирования является условным и соответствует определенному спектру действий внутри инструментального средства по реализации УПСЭИ ЕПС и её возможностей по работе с ИЕХ. Если продолжить эту классификацию в сторону "увеличения" возможностей программиста по разработке своих систем до уровня операционной системы, то мы получим машинно-ориентированный язык из нулей и единиц. А если в сторону расширения возможностей пользователя, то получим абстрактную систему без программирования (СБП-систему), которая должна быть предельным случаем автоматизации ([4], с. 13).

Сразу оговоримся, что под СБП-системой здесь и далее автор будет понимать некий гипотетический программный продукт, носящий на себе черты многих инструментальных средств, к которым относят CASE-системы, PDM-системы и другие подобные им по реализации возможностей программиста системы. 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 СП.

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) описания 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 работы пользователя при работе с данными на различных уровнях (копирование файлов, калькулятор, работа с дисками и т.п.).

Легко заметить корреляцию с УПСЭИ. Кроме того, в состав СБП-системы могут входить другие средства, ориентированные как на пользователя, так и на программиста (например, стандартные примеры, Online-учебник и т.п.).

СБП-система может быть полной, если будет включать в себя все перечисленные выше компоненты, и частично реализованной, то есть предоставлять проектировщику только часть данных функций. Например, СП FoxPro 2.0 под DOS имела в своем арсенале поддержку многих из этих компонент и была наиболее близка в указанном смысле к полной СБП-системе. Автоматизация программирования поддерживалась в этом средстве через создание шаблонов, по которым специальная утилита генерирует исходные тексты программ. Как всякое универсальное средство программирования, FoxPro этой версии не имел заранее определенного шаблона, который годился бы в качестве стандартного для синтеза многих ЕПС в АСУП любой предметной области и любого класса задач. Точнее говоря, это средство имело несколько шаблонов, каждый из которых подходит для некоторого небольшого круга наиболее простых задач или реализовывал какой-то один класс задач, отличаясь при этом оформлением диалога с пользователем. Средство предлагало на выбор несколько шаблонов, которые могут быть использованы для этих целей по выбору программиста. Разумеется, для реальных АРМ возможностей, заложенных в этих шаблонах, недостаточно, поэтому программисту нужно было их дорабатывать. Такие шаблоны были прообразом систем проектирования программных продуктов.

Доработка шаблонов всякий раз выполнялась по-разному для различных ЕПС, создаваемых даже одним программистом. При возрастании числа реализованных ЕПС возрастало и число шаблонов. Со временем, когда число шаблонов стало больше 10, возникла необходимость подробного описания того, как эти шаблоны составлены. Эта информация требовалась программисту, так как исходя из накапливаемой информации, он должен понять как использовать в дальнейшем свой опыт.

Ситуация совершенно не меняется при использовании других программных продуктов. Excel и Access, в том виде, в котором они сейчас предлагаются в качестве инструментального средства создания приложений, позволяют при определенном знании языка диалога, структур данных, представлении о базах данных и их вспомогательных файлах написать более или менее снос-

ную "программу" самому пользователю без участия программиста. Она могла бы автоматизировать какую-то часть деятельности пользователя. Расширение подобного продукта и его доработка силами пользователя весьма проблематична, так как начинает требовать от занимающегося данным видом деятельности слишком большого количества технических знаний, мешающих ему в основной деятельности.

Перейдя к обобщению понятия СБП-системы, следует сказать, что степень её реализации зависит от того, насколько полно её разработчики представили себе процесс программирования ЕПС, а в рамках АСУП крупного предприятия и процесс управления синтезом многих приложений при их накоплении. Поэтому можно ввести понятие абсолютно полной СБП-системы, которая включает в себя также инструменты работы с приложениями и определяет порядок "разграничения полномочий" пользователя ЕПС и создателя ЕПС. Фактически, абсолютно полная СБП-система и есть тот ключ, который позволяет значительно снижать трудозатраты на реализацию программных продуктов (или большой группы ЕПС в рамках АСУП).

Термин "абсолютно полная СБП-система" введен в статье, но не реализован на практике и представляет собой некое гипотетическое обобщение. Однако, по степени приближения к данному идеалу и количеству компонент и функций, реализованных в гипотетическом или конкретном средстве программирования, можно судить о степени реализации в нем абсолютно полной СБП-системы. Помимо этого можно заметить, что любой программист, создавая библиотеки подпрограмм или компонентов реализации СПО в ПП ЕПС, интуитивно стремится реализовать в них ту функциональность, которая будет соответствовать абсолютно полной СБП-системе.

Прежде чем проводить дальнейшие рассуждения на эту тему, введем еще несколько понятий, связанных с процессом программирования и программистом. Инструментальное средство – система программирования или иной программный продукт, посредством которого программист создает СПО или иным образом реализует требования технического задания в отношении решения задачи автоматизации единичной конкретной предметной области. Это понятие неоднократно употреблялось по тексту выше, однако строго определено только здесь. Это вызвано тем, что в технической литературе по данной тематике это понятие считается чем-то само собой разумеющимся и относящимся к ЯВУ или ЯП. Это не совсем язык и не только язык, а еще и набор инструментов программиста (редактор текста программ, отладчик, сборщик файлов исходного кода и др.). Однако в связи с появлением CASE-систем, PDM-систем и других им подобных по реализации возможностей программиста систем рамки данного понятия расширились.

Инструментарий программиста – набор программ, компонентов, файлов описания свойств, переменных и заголовков, используемый им для реализации ПП ЕПС более одного раза. Это понятие есть отношение, построенное на множествах представлений программиста о технике (или техниках) программирования, его знаний о конкретной версии конкретного инструменталь-

ного средства, характеристических моделей разработанных им ПП ЕПС, понятий эстетических предпочтений и профессионального образования. Столь сложное понятие с точки зрения формального анализа тяжело моделировать и анализировать, но очевидно, что основным параметром этого понятия является программист. Реализовывался инструментарий программиста при разных подходах к синтезу СПО по-разному. Чрезвычайно распространенным долгое время был метод создания библиотек сначала подпрограмм, а затем и компонентов. С появлением инженерии предметной области и аспектно-ориентированного программирования [5] стали появляться библиотеки реализации характеристических доменных моделей и библиотеки аспектов. Помимо этого широко развивается направление, связанное с применением библиотек конфигураций. Насколько эффективно их создание с точки зрения экономии трудозатрат, быстродействии и использования ресурсов, сейчас сложно сказать, так как это достаточно новые технологии, ждущие своего аналитика и конструктивного критика.

Тем не менее, исходя из введенного понятия инструментария, можно ввести понятие ***"идеального инструментария"*** программиста. Это инструментарий, позволяющий программисту создавать группы (семейства, линейки) ПП ЕПС без его доработки. Идеальный инструментарий, как и идеальный газ в природе не существует. Поэтому программисты обычно работают с реальными инструментариями. Они создают их при работе над проектами и, как было сказано выше, интуитивно стараются привнести в свою работу функциональность СБП-систем. Подобно семантическим элементам интерфейса, введенным в предыдущем разделе для отделения реализуемой смысловой части от инструментального средства, детали (компоненты, модули, подпрограммы) реального и идеального инструментариев могут не зависеть от инструментальных средств. Подобная данной идеология отчасти сейчас вводится благодаря появлению Microsoft технологий "***NET***" и J2EE/EJB. В этих системах программисты отделяют понятие интерфейса объекта от его реализации и вводят стандарты на интерфейс и названия объектов.

В состав своего реального инструментария программист стремится внести как можно больше разнообразных элементов, сравнить которые можно с радиодеталями и схемотехническими изделиями. Такие "кирпичи" работают в определенном диапазоне: они имеют определенный тип передаваемых параметров, набор свойств, набор возвращаемых значений и переменных для хранения информации. При увеличении "порядка" сложности (в экспоненциальном смысле) или вложенности такого компонента (программы) возможно получить один из компонентов СБП-системы в "чистом" виде. Но далеко не в каждом инструментарии доходит до этого.

Обычно программист останавливается на частичной реализации функциональности компонентов СБП-системы. Поэтому другим важным определением в этой области можно считать ***функциональность компонента СБП-системы***, под которой следует подразумевать наборы свойств и методов этого компонента, полностью определяющие его как независимую и самодостаточ-

ную часть СПО. Функциональность идеального компонента СБП-системы подразумевает наличие максимально возможного реализованного в нем количества функций, характерных для данного направления (аспекта, модуля, группы связанных функций) СПО. Реальные компоненты СБП-системы реализуют некоторые или многие из предусмотренных к реализации в рамках данного аспекта функций.

Если говорить о технике синтеза СПО и хранения информации для него, то каждая из описанных частей СБП-системы создает одну (несколько) ТБД, которая хранит информацию об одном или нескольких ЕПС. Такую информацию обычно принято называть метаинформацией (или конфигурацией). Метаинформация хранит данные о таблицах БД, их составе, ограничениях, индексах, связях и т.п. [6]. Но в случае с СБП-системой это не совсем так, поскольку система поддержки приложений должна быть на один уровень выше метаинформации и хранить данные о наборах системных таблиц, касающихся разных ПП ЕПС.

Разумеется, для большего удобства программиста и пользователя необходима реализация ЕПС в виде исполняемого в ОС файла, который копировался бы в нужный каталог вместе с БД пользователя. При этом накопление приложений становится неудобным, так как каждое из них, с одной стороны, требует много места на ПЭВМ разработчика, а с другой – "запутывает" или непомерно увеличивает системную информацию. Такая информация требуется различным ЕПС, в том числе и СБП-системам, при работе на одной ПЭВМ. Выходом из такой ситуации для многих коммерческих инструментальных средств стало формирование файлов конфигурации [7, 8].

Следует также дополнительно остановиться на понятии уровня детализации. Формирование функциональных частей СПО ЕПС при помощи приведенных выше частей полной СБП-системы может включать в себя различной глубины диалоги с разработчиком. Целью этих диалогов является реализация ПП ЕПС. Самым простым количественным критерием уровня детализации в данном случае может служить максимальное количество задаваемых системой вопросов (предлагаемых запросов по меню или вопросительных реакций СБП-системы). Более сложным критерием можно назвать "продуманность" действий СБП-системы при создании ЕПС. Часто широко продаваемые на рынке программные продукты не могут обеспечить нужный уровень детализации для данного класса задач и выходят из положения, предлагая пользователям своих систем наборы шаблонов, программ, примеры приложений. Для пользователей-непрограммистов данная ситуация не слишком удобна, так как требует внимательного изучения инструментального средства.

Для программистов трудности при таком подходе вызваны тем, что при синтезе приложений с помощью таких программных продуктов как СБП-системы возникает проблема накопления приложений и их согласования между собой. Часто эта проблема бывает неразрешима в рамках данного инструмента программирования и поэтому требует от программиста применения организационных методов работы с созданным СПО, которые он обычно описывает в инструкциях (руководствах) по эксплуатации.

Это могут быть:

- ведение журналов учёта изменений системной информации, вызванных настройкой на конкретный тип операционной системы или аппаратных средств;
- создание макетных участков;
- изменения в настройках (БД СБП-системы) в связи с требованиями конкретного пользователя и отражение этих изменений в каких-то документах, другие виды подобных мероприятий;
- установка на разных компьютерах разных приложений или разных их копий для различных пользователей.

Поэтому помимо описанного выше состава компонент СБП-системы, используемой в качестве базового инструмента программирования, в её состав должен быть включен компонент работы с группами приложений, который должен включать в себя следующие менее обобщенные компоненты:

- описания свойств предметной области по отношению к конкретному приложению (направление деятельности пользователя и характер использования ПЭВМ);
- называемый "tools-kid", используемый данным классом задач в качестве вспомогательного средства;
- учётная информация по использованию и доработкам задач данного класса, входящих в создаваемую группу.

Самые трудно описываемые места, связанные с доменными моделями, могут быть записаны в виде одной или нескольких команд (формул), а остальные части того, что ранее было программой, записывается в виде таблиц (БД) при помощи удобных средств ввода (меню, таблиц или др.). Разумеется, созданный при помощи данных средств программирования продукт становится БД (группой БД), а создание такого средства программирования превращается в программирование баз данных. Традиционные языковые средства СУБД имеют наиболее общую направленность, ориентированную на максимальную универсальность круга тех ЕПС, которые будут разрабатываться с их помощью.

Если говорить об общей парадигме процесса эволюции взглядов на развитие программирования, то появление признаков СБП-систем можно представить так. В начале 80-х годов в США появилась концепция программирования, согласно которой данные были отделены от программ и возникли объекты программирования, называемые базами данных. В дальнейшем эта концепция претерпела изменения, благодаря которым программы превращаются в БД, а данные опять соединяются с программами в одном месте перед исполнением последних. Так во всех сетевых БД есть инструмент, называемый ***хранимыми процедурами***, а в приложениях на базе Access программные модули, формы ввода и отчеты включены в состав единого файла, который принято называть БД. Процесс повторился на более высоком уровне.

Итак, СБП-система может считаться наиболее близким приближением к идеальному инструментарию программиста. Следовательно, её наличие и "одинаковость реализации" в различных инструментальных средствах позволит увеличить производительность труда программистов при синтезе СПО

для АСУП её спецификации на уровне компонент, их частей, свойств и методов, а также событий, реакцию на которые можно предусмотреть, и возможно позиционировать как набор смысловых понятий, не зависящих от инструментального средства.

Не обладая стандартом, не зависящим от средства программирования, и принимая FoxPro за вариант стандарта реализации абсолютно полной СБП-системы, можно утверждать, что FoxPro версии 2.0, реализует полную СБП-систему не более чем на 63,4 % или 7 функций из 11. Абсолютно полную СБП-систему он не реализует.

В MS Access многие компоненты СБП-системы были реализованы. Но данный продукт позиционировался на рынке инструментальных средств как хороший продукт для начинающих. А в некоторых источниках технической литературы его вообще называют универсальным продуктом на все времена [6]. Если сравнивать его с другими используемыми для синтеза СПО инструментальными средствами, которые программисты используют несколько чаще остальных, то у него есть ряд явных недостатков, связанных, прежде всего, с инструментом вызова объектов БД из модулей и процедур (User Define Procedures) и со своим определением работы операторов языка SQL при определении запросов.

Самым простым способом синтеза АРМ посредством СБП-системы следует считать разработку шаблона для генератора исходного кода, язык которого интерпретируется средством генерации исходного кода ЕПС, поставляемым фирмой-разработчиком вместе с инструментальным средством. Он используется как входная информация, к которой на определенном этапе подключаются отдельно разработанные экранные формы, базы данных и отчеты. Очевидно, это наименее рекомендуемый сейчас способ создания СБП-системы.

Более сложным вариантом может служить доработанный для различных классов задач автоматизации набор расчётов, экранных форм, отчетов и меню, который незначительно меняется от одной к другой. Такой вариант шаблона может носить название базового. Базовой моделью СПО АРМ для данного инструмента программирования (СБП-системы) в данном случае является исходный текст на ЯП, содержимое файлов описания экранов, отчетов, расчётов и запросов (содержание БД по данному классу задач). Для инструментальных средств, не предусматривающих создание кода, такой моделью может служить конфигурация.

Наконец, еще более продвинутым вариантом мог бы стать хранящийся и изменяемый в диалоге при необходимости набор условий или правил формирования экранных форм, правил создания отчетов и расчётов, меню, порядка ведения архива и обмена данными и т.п. Этот набор, реализованный подобно метаинформации в БД, мог бы быть группой её таблиц, не связанных с конкретным инструментальным средством и классом задач. Для данного класса задач может быть создан некий "идеологический" tools-kid, выполняющий роль основы для синтеза конкретного вспомогательного пакета на любом из существующих и вновь появившихся ЯП (инструментария программиста). Точкой от-

счета в построении такого набора правил могли бы стать структуры данных, описанные как ИЕХ, и диалоговые средства, позволяющие заполнять созданные на их основе таблицы нужной информацией.

Таким образом, универсальное обобщение методов синтеза СПО в виде набора компонентов может быть выполнено при помощи описанной выше СБП-системы. С одной стороны она является средством синтеза СПО, а с другой – разрабатываемым программным продуктом и инструментарием программиста. На основе предложенной модели построения исходного кода возможно введение понятия идеального инструментария программиста, как объекта, позволяющего реализовывать семейства ЕПС (линейки программных продуктов), предельным случаем которого может служить абсолютно полная СБП-система.

Литература

1. ***ISO/IEC*** 12207: 1995-08-01. Центр Информационных Технологий.

2. ***ГОСТ*** Р ИСО/МЭК 12207-99. Информационная технология. Процессы жизненного цикла программных средств.

3. ***Кобринский Б.А.*** К вопросу о формальном отражении образного мышления и интуиции специалиста слабо структурированной предметной области // Новости искусственного интеллекта. 1998. № 3. С. 64-76.

4. ***Ильин В.Д.*** Система порождения программ. М.: Наука, 1989.

5. ***Чарнецки К., Айзенкер У.*** Порождающее программирование. Методы, инструменты, применение / пер. с англ. СПб.: Питер, 2005.

6. ***Крёнке Д.*** Теория и практика построения баз данных / пер. с англ. СПб.: Питер, 2005.

7. ***1С:Предприятие*** версия 7.7, Конфигурирование и администрирование, часть 1-2, фирма "1С". М., 1999. http://www.1c.ru.

8. 1***С:Предприятие*** 7.7, Конфигурирование и программирование. Базовые объекты // Курс дистанционного обучения. ООО "1С-Учебный центр", 2004.