УДК 004.424, 004.415

В.Г. Колесник

DS-ТЕОРИЯ. НАУЧНЫЕ АСПЕКТЫ И ПЕРСПЕКТИВЫ РАЗВИТИЯ

В работе представлена теория схем декомпозиции как научная теория. Описаны ее атрибуты – парадигма, поле исследований, цель исследований, основная задача, метод решения, основная теоретическая модель – схема декомпозиции. Утилитарная, практическая цель теории – предложить механизм генерации прикладных алгоритмов (не машинного кода). Показано, что схема декомпозиции как описание заменяет описание алгоритма и при этом остается декларативным в противоположность описанию алгоритма, как императивному. При этом схема декомпозиции есть описание, исходное для генерации алгоритмов. Описаны виды схем декомпозиции и операции над ними. Описаны алгоритмически релевантные факторы, которые следует учитывать при генерации алгоритмов для того, чтобы алгоритмы становились реальными. В работе описана работа по формализации и математическому описанию явлений и объектов теории схем декомпозиции. Предложен механизм контроля выводов и результатов теории. В работе также описано направление развития теории схем декомпозиции. Приоритетное направление соотносится с созданием и развитием системы емких понятий и абстракций. С точки зрения практического применения теории схем декомпозиции, предложен подход подобный тому, что есть в машиностроении – развитие и продвижение передовых технологий по запросу, по потребности – как результат целенаправленных исследований.

Ключевые слова: алгоритм, анализ, генерация, данные, декомпозиция, наука, программная инженерия, программирование, процесс, структура, синтез.

Введение

Несмотря на то, что в сфере программной инженерии накоплено множество фактов, разработаны методы, предложены обобщения и абстракции, – теории общего плана не существует. *DS*-теория, описанная в [1–5] – это попытка построить систематическую теорию для программной инженерии (для технологии программирования). В настоящей работе обсуждаются различные аспекты *DS*-теории как научной теории, развивающейся на основе единой парадигмы. Описаны факторы, являющиеся препятствием к применению *DS*-теории в практике. Принимая их во внимание, теоретические выводы можно применять в практике реального практического программирования. Проводится сравнение программирования с применением *DS*-теории с одной стороны и машиностроительным производством с другой. На основании этого сравнения обсуждается возможный эффект в сокращении издержек – именно то, что может дать индустрии программирования *DS*-теория как научная теория.

Атрибуты научной теории

Поле исследований. *DS*-теория объектом своего внимания имеет алгоритмы программ (или программных комплексов) во всех прикладных областях. Прежде всего, исследуется управление – конструкции из циклов, условий, подпрограмм. *DS*-теория занимается исследованием тех факторов, что порождают управление, структуру алгоритмических конструкций. Исследуются причины, из-за которых растет сложность алгоритмических конструкций.

Парадигма. Явно или неявно, в самом общем виде в разработке программного обеспечения существует одна теоретическая модель – некое подобие машины Тьюринга. С точки зрения этой модели, обрабатываемые данные являются абстрактными и безликими. А это значит, что основные элементарные операции над данными являются абстрактными, примитивными и бедными. Эта модель с точки зрения программной инженерии не формализована, но на ее основе построены текстовые процессоры, СУБД, компиляторы и

интерпретаторы, на ее принципах построено объектно-ориентированное программирование [6] и посредством него эта модель массово внедряется в разрабатываемые программы. В настоящее время эта модель превратилась в методологическое препятствие в развитии программной инженерии.

В *DS*-теории вместо модели "Машина Тьюринга" используется модель "Формирование знаний". Принципиальная ключевая исходная идея *DS*-теории (парадигма) звучит так: "В компьютере производятся новые знания". Эта парадигма принимается в *DS*-теории как аксиома, как исходное положение и противопоставляется другой парадигме: "В компьютере происходит обработка информации". Развитие основной идеи в следующем: "Знания имеют свою внутреннюю структуру и свой механизм их формирования – процедуру декомпозиции". И далее: "Алгоритм и программа в компьютере являются отражением процедуры декомпозиции".

Процедура декомпозиции известна и интуитивно понятна практически любому исследователю, так как это один из мыслительных инструментов исследований. Кроме того, схема декомпозиции – это то, что очень похоже на структурный анализ. Но в *DS*-теории акценты устанавливаются на процедуре формирования знаний, а не на наборе декомпонуемых конструкций и механизме управления ими.

Цель исследований. Главной целью *DS*-теории есть создание средств эффективного управления сложными программными проектами – борьба со сложностью. Понимая, что для многих проектов сложность – это естественное свойство, – это природа проекта и уменьшить ее не представляется возможным, то *DS*-теория нацелена на то, чтобы разработать средства для управления сложными проектами.

Производными целями *DS*-теории, которые могут быть достигнуты, есть те же цели, что определились как основные в индустрии программного обеспечения:

- повысить качество программ. Более конкретно – уменьшить количество ошибок на порядок;
- обеспечить сокращение времени программирования. Для этого уйти от программирования с использованием алгоритмических языков строчного типа к языкам графического типа;
- обеспечить сокращение издержек на разработку и сопровождение программ в разы.

Основная задача. Поставленные выше цели *DS*-теория предполагает с помощью решения основной задачи. Такая основная задача – это проектирование схемы получения новых знаний об объекте познавательной схемы (ПС). Построение этой схемы подразумевает существование трех обстоятельств. Первое то, что ПС будет записана строго формально. Второе – как правило, ПС будет сложной и структурированной, и будет состоять из более простых схем как компонент. И третье то, что ПС будет содержать повторяющиеся расчетные действия.

В рамках *DS*-теории дополнительно предпринимаются шаги для решения еще некоторых задач. Разрабатываются средства компактного описания алгоритмов и программ. А для этого ведется поиск емких понятий – основное продуктивное средство *DS*-теории. Компактное средство описания программ подразумевает последующую разработку графических визуальных средств описания, – это основная предпосылка для создания эффективной графики.

Метод решения. Для того, чтобы представить ПС в формальном виде как универсальную модель, используется формальная схема декомпозиции. Процедура построения включает эмпирические исследования ПС в прикладной области. В простейшем случае проектирование будет заключаться в том, чтобы только записать формально ПС (или, то же самое – решение прикладной задачи), которая реально существует в ручном расчете, либо в автоматизированном расчете, но с использование менее продуктивных технических средств. Возможно, что ПС уже записана и реализована в виде программы, но ее необходимо расширить функционально. Самый сложный вариант

– ПС не существует и ее необходимо спроектировать сначала при отсутствии прототипов.

Завершением решения будет преобразование ПС в реальный алгоритм, представленный на алгоритмическом языке или как исполняемая программа в машинном коде. Собственно, это есть генерация алгоритма на основе формального описания схемы декомпозиции.

Основная теоретическая модель. Основной инструмент теории или основная теоретическая модель – схема декомпозиции[1]. Простейшее описание схемы декомпозиции [1] включает дерево полной схемы декомпозиции (DPS), где для каждого узла K_i должен быть определен кортеж описания узла $KR_i = (T_i, A^a_i, A^s_i, DM_i)$, который включает:

T_i – перечень типов свойств.

A^a_i – аналитические алгоритмические зависимости.

A^s_i – синтетические алгоритмические зависимости (САЗ).

DM_i – частную схему декомпозиции (ЧСД).

K_i – идентификатор узла, где индекс отражает номер узла при обходе дерева сверху вниз слева направо. Аналитические и синтетические зависимости имеют общее название – алгоритмическая зависимость (А-зависимость). ЧСД предписывает способ разделения объекта или его части только на непосредственно составляющие их компоненты. ЧСД есть описание одного шага декомпозиции и отличается от полной схемы, тем, что последняя описывает полную декомпозицию. Полная схема декомпозиции описывает части объекта и его свойства сколь угодно глубокого деления или, с точки зрения дерева декомпозиции, может содержать произвольное количество уровней. Символьное описание схемы декомпозиции имеет вид:

$$DPS=\{K_0[KR_0]\ (K_1[KR_1]\ (\ldots), K_r[KR_r]\ (\ldots), \ldots, K_z[KR_z]\ (\ldots))\}.$$

Кортежи листьев не содержат ЧСД и САЗ. Схема декомпозиции изображается древовидным графом, в котором нагружены и ребра, и узлы. Ребра нагружены в том смысле, что каждая ЧСД и САЗ соотносится с одним конкретным ребром. А-зависимости представляются символьными строками, которые реализуют аналитические формулы, что может повлечь дополнительные грамматические конструкции строчного типа. Следует заметить, что А-зависимость более широкое понятие, чем формула и предполагает любой способ описания зависимости – аналитический, словесный, табличный и т. п.

Описание схемы декомпозиции – это описание статического объекта. DPS – это дерево, а А-зависимости и ЧСД – это описание отношений. Кроме этого при более детальном описании узлов схемы декомпозиции могут быть использованы опции и параметры. Схема декомпозиции есть симбиоз всех этих видов описаний. При описании схемы декомпозиции отсутствуют любые императивные утверждения.

Более детальное описание всего, что связано с узлом, это не просто кортеж как перечисление, а некоторая конструкция – алгоритмическая конструкция узла (АКУ). Ее компоненты – А-зависимости и ЧСД сочленены в единое целое. Структура АКУ зависит от вида и количества компонент и способа их сочленения. Тот порядок, в котором сочленены компоненты АКУ и вид сочленений определяют предварительный порядок расчета – реализации А-зависимостей. Точный порядок расчета будет зависеть от многих факторов, которые задаются опциями и параметрами. АКУ более развитое понятие чем узел с кортежем.

Для реализации схемы декомпозиции (СД) в [1] предложен канонический алгоритм (КА) – это универсальный алгоритм, не зависящий от структуры DPS. Он может содержать произвольное количество узлов, ветвей, уровней, а у любого узла

[1] В научной среде есть понимание того, что для развития программной инженерии как научной дисциплины необходима теоретическая модель как некая обобщающая родовая конструкция для как можно большей совокупности алгоритмов и алгоритмических конструкций [7, 8].

может быть произвольное количество исходящих ветвей. Универсальность его в следующем:

- с каждым не конечным узлом соотносится набор определенных видов компонент (с листьями соотносится меньший набор видов компонент);
- задан определенный порядок реализации А-зависимостей – обход дерева сверху вниз слева направо.

КА как порядок выполнения некоторых действий не зависит от прикладной области. Кроме того, каким бы ни были DPS и А-зависимости, которые соотносятся с узлами DPS, в результате выполнения должен быть замкнут контур синтеза (КС) [1]. Это является критерием правильности КА.

АКУ, несмотря на то, что набор видов компонент определен и ограничен, не универсален. Структура АКУ зависит от вида, количества и взаимодействия А-зависимостей, от того как сочленены компоненты АКУ. Все перечисленные факторы порождают многообразие АКУ.

ЧСД вместе с понятиями “КА”, “Р-данное”, “А-зависимость” и “АКУ” представляет кластер понятий. В таком смысле ЧСД с концептуальной точки зрения есть базовое понятие подобно тому, как базовым есть понятия “класс” и “объект” в объектно-ориентированном программировании. ЧСД как базовое понятие имеет методическую ценность, поскольку на общем уровне указывает направление проектных действий. В плане методического эффекта ЧСД – это предпосылка для создания системы программирования на основе *DS*-теории существенно превосходящей ООП в плане производительности.

Практическое использование СД. Любая ПС или любая задача в прикладной области рассматривается с точки зрения того, является ли она схемой декомпозиции. То есть, СД рассматривается как универсальный шаблон.

Если ПС уже существует в каком-либо виде, в виде программы или ручного расчета, то анализ заключается в том, чтобы увидеть, может ли быть представлена эта программа или расчет в виде СД. Если это возможно то, первый этап выполнен. Далее предстоит превратить СД в работающую программу. Если ПС не существует ни в каком виде и ее следует проектировать, то она проектируется и представляется в виде СД. В процессе анализа может оказаться, что ПС может быть представлена в виде СД только частично – только как фрагмент СД и ее необходимо дополнять ручным или автоматизированным расчетом. В практике это обычное явление.

Как правило, и существующие ПС, и такие, что проектируются сначала, в качестве результата содержат более одного данного. Каждое из искомых данных имеет уникальную схему расчета. В таком случае в СД совмещаются, насколько возможно, схемы расчета для всех искомых данных. То есть, конечная СД является результатом синтеза более одной СД.

Далее формальное описание СД используется как заказ на генерацию. Возможность генерации алгоритмов и программ есть неотъемлемой частью *DS*-теории.

Объект исследований

СД – это универсальное средство в руках человека и применима к исследованиям и решению различных задач практически во всех сферах деятельности человека. Тем не менее, между СД в различных прикладных областях есть различия. Эти различия порождают различия в прикладных программах и алгоритмах. Дополнительный источник многообразия в алгоритмах исполняемых программ – это вычислительная среда, в которой данные программы работают.

Виды СД и ее компоненты. Декомпозиция – это один из основных методов, применяемых человеком для исследования и преобразования окружающей действительности. Человек, выполняя декомпозицию, преследует следующие цели:

- исследование существующего объекта. Расчленяя (разбирая, раскладывая) объект или вычленяя компоненты объекта, человек намерен узнать новые неизвестные ему ранее, свойства, новые знания об объекте. Это может быть его конечной целью;

- создание нового объекта. Проектирование объекта мысленно или с помощью каких-либо средств, предполагает создание проекта. Последовательно могут создаваться эскизный проект, технический, рабочий. Но этот процесс также есть и декомпозиция. Создание проекта есть мысленная композиция;
- преобразование существующего объекта. Преобразование может быть вызвано необходимостью ремонта или модернизации объекта. В результате преобразования предполагается, что объект получит новые свойства. Преобразование предполагает разборку или расчленение объекта, а затем удаление или замена некоторых старых компонент или (и) добавление новых. В результате этого преобразования объект получает новые свойства или новые функции, что есть то же самое. В этом процессе есть и декомпозиция, и композиция.

Эти три цели из-за сочетания различных ЧСД при их достижении порождают алгоритмическое многообразие. Поэтому одной из функций *DS*-теории есть исследование и типизация как полных схем декомпозиции, так и ЧСД.

Декомпозиции может быть подвергнуто все, куда устремляется осмысленный познавательный взгляд человека. Декомпозиция предшествует всему, что намерен произвести человек, для последующего использования. Поэтому объектов декомпозиции в реальном мире весьма много. Много также различных видов и аспектов декомпозиции. Ей могут подвергаться:

- объекты живой или неживой природы, или продукт человеческого труда (артефакт);
- процесс, как динамический объект или предмет (организм), как статический объект;
- объекты начиная от микромира до объектов макромира;
- объекты, которые подвергаются декомпозиции в соответствии с заранее составленным планом или объекты, схема декомпозиции которых, может меняться в процессе исследования;
- объекты техники и объекты искусства;
- объекты, на которые воздействуют или объекты, за которыми наблюдают без воздействия;
- объекты, которые существуют реально или только в сознании исследователя;
- процесс реальный или материальный и процесс, полностью моделируемый в компьютере (различного вида тренажерные программы или динамические игры).

Декомпозиция может выполняться с помощью технических средств полностью или частичным их использованием. Также декомпозиция может учитывать результаты предшествующих шагов или изменение свойств объекта или не учитывать.

Объектами декомпозиции могут быть информационные объекты, полностью размещаемые в компьютере. Схемы декомпозиции и композиции таких объектов – это прототипы таких программ как браузеры, функционально-развитые редакторы текстов или графических изображений, издательские системы, системы САПР, игровые программы, различные модели. Из-за их сложности их специфика такова, что порой они до конца не бывают отлажены в принципе.

Все это многообразие объектов и видов декомпозиции порождает многообразие ЧСД и САЗ. Однако *DS*-теория акцентирует внимание на алгоритмических аспектах ЧСД и САЗ. Важным есть то, что в процессе представления в виде алгоритмов порождает циклы, условия – то, что порождает управление в алгоритме.

Кроме того, использование компьютеров при выполнении декомпозиции порождает дополнительные вариации в алгоритмах процедур декомпозиции. С абстрактной точки зрения основной работой, которая выполняется в процессе декомпозиции, есть работа с данными (значения свойств) – это сбор, фиксация, генерация, вывод данных. Более детально это может выглядеть так:

- данные расположены в файлах или в базах данных. В процессе декомпо-

зиции их только вводят в компьютер для обработки;

- данные снимают с датчиков и вводят в компьютер в реальном режиме времени. Данные вводятся либо по мере поступления, либо в соответствии с некоторым регламентом;
- декомпозиция моделируется в компьютере. Данные при этом формируются в соответствии с некоторой функциональной или более того, в соответствии с алгоритмической зависимостью и после выводятся на управляющие устройства, либо в базы данных.

Возможны сочетания этих вариантов. Эти три варианта работы алгоритмов схем декомпозиции и их возможные сочетания тоже являются основаниями для исследования и типизации ЧСД.

Разнообразие СД есть еще и в том, что различается степень участия человека (и компьютера) в реализации расчетов СД. Диапазон взаимного участия в комплексе человек-компьютер распространяется от полностью автоматического расчета, до полностью человеческого расчета с использованием калькулятора на экране монитора компьютера или просто чтение текста на планшете. Полностью автоматизированный расчет – это значит, то вся СД полностью вычисляется в компьютере. Или что, то же самое, КС полностью вычисляется в компьютере. В этом отношении есть вариации двух видов:

- СД, начиная с корневого узла и до какого-то уровня, вычисляется в компьютере. Вне компьютера вычисляется часть СД, начиная с узлов определенного уровня и ниже. По разным ветвям уровни, где завершается автоматизированный расчет, могут быть различными. Для такого типа СД необходимо обеспечить вывод данных для последующего ручного расчета;
- СД, начиная с корневого узла и до какого-то уровня, вычисляется вне компьютера. В компьютере вычисляется часть СД, начиная с узла следующего уровня и ниже. Для такого типа СД необходимо обеспечить ввод данных для последующего ручного расчета.

Процесс расчета СД может быть разделенным во времени. Данные частично сформированы ранее и хранятся в сети, в базах данных или в Интернете – это первая часть расчета, а потом эти данные используются для расчетов – это вторая часть схемы.

Процесс расчета может инициироваться человеком или компьютер может быть в режиме ожидания, а расчет в соответствии с СД инициируется неким процессом. СД с подобными вариантами запуска расчета могут быть синтезированы в одно целое.

Таким образом, СД и подвергаемые ей объекты есть объектами исследований в *DS*-теории. Более точно, объектом исследований *DS*-теории есть алгоритмическое представление СД.

Операции над СД. Существует несколько операций над СД[2]: дополнение, усечение, сцепление и три вида синтеза. Синтез как дополнение А-зависимостей; синтез как совмещение СД и синтез схем декомпозиции и композиции.

1. Дополнение. Данная операция означает, что в процессе исследования объекта есть необходимость сделать более глубокую декомпозицию. К существующим компонентам объекта применяется более глубокая декомпозиция. С точки зрения дерева, с помощью которого представлена СД, это значит, что появилась еще одна ветвь – дерево дополнено еще одной ветвью. Хотя дерево исходной схемы может быть дополнено схемой, у которой ветви более чем на одном уровне.

2. Усечение. Эта операция означает, что в процессе исследования объекта исчезла необходимость делать глубокую декомпозицию. Дерево СД стало короче на одну или более ветвей.

3. Сцепление. Данная операция значит, что в процессе исследования объекта необходимо выполнить дополнительный расчет (один или более). С точки зрения дерева схемы декомпозиции это значит, что добавляются дополнительные ветви на любом узле.

[2] Первые три из перечисленных операций и один из видов синтеза описаны в [1].

4. Синтез как добавление А-зависимостей. Эта операция означает, что к объекту необходимо применить еще одну СД с механизмом декомпозиции, который был применен ранее и выполнить расчет в соответствии с дополнительной А-зависимостью. Дерево исходной СД в этом случае не меняется.

5. Синтез как совмещение СД. Данная операция означает, что к объекту необходимо применить более одной СД с различными механизмами декомпозиции. Механизмы декомпозиции различаются тем, что делят объект на части, размеры которых не совпадают. То есть, для каждой схемы уникальный размер отделяемой части. Хотя размеры частей могут различаться и для одной схемы.

6. Синтез схем декомпозиции и схем композиции. Как правило, работающая программа одновременно управляет процессом декомпозиции и процессом синтеза или композиции нового объекта. Так как любой процесс (и декомпозиции и композиции) может быть представлен СД (и, соответственно, деревом), то конечный работающий в компьютере процесс есть результат синтеза СД (и, соответственно, синтеза деревьев). Если быть предельно точным, то простейший алгоритм – чтение файла по записям с одновременным выводом записей в другой файл тоже есть результат синтеза двух схем. Одна из этих схем изображает чтение файла, а другая – вывод. Но более важным является синтез СД, которые, будучи результатами синтеза, уже являются весьма сложными многоуровневыми деревьями. Алгоритмы, строящиеся традиционными методами для выполнения подобных функций, которые могут быть выполнены подобными СД, порой с трудом поддаются осмыслению и контролю[3].

[3] По мнению многих программистов, суть программирования – построение сложных конструкций из условных операторов, операторов цикла, – построение конструкций управления и их завершенная отладка. Но ни одна из предложенных ранее систем программирования или методологий не пыталась системно решить задачу генерации управления. DS-теория предпринимает такую попытку.

Эффективное использование операций над СД предполагает их типизацию и изучение с точки зрения их алгоритмической природы. Возможны ситуации, когда в каких-то случаях ЧСД или СД необходимо совмещать (или синтезировать) с САЗ. Это есть дополнительная причина исследования и типизации ЧСД и САЗ.

Синтез алгоритмов. Операции над СД уже есть синтез алгоритмов и, соответственно, программ. После того как СД синтезирована, *DS*-теория обеспечивает ее преобразование в алгоритм и программу на алгоритмическом языке. Дальше нет препятствий для того, чтобы преобразовать эту программу в исполняемую программу.

СД имеет алгоритмическую природу и именно поэтому один из видов ее описания, в котором представляется СД есть КА. Одновременно КА – это заказ на основании, которого производится синтез конечного алгоритма и программы. Синтаксические структуры КА и программы, и взаимосвязь их синтаксических компонент показаны на рис. 1.

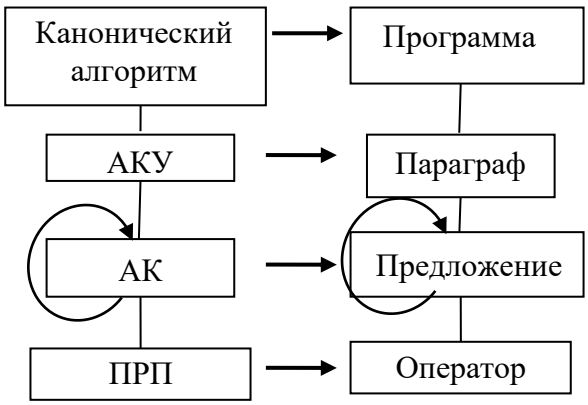

Рис. 1. Взаимодействие синтаксических конструкций СД и конструкций алгоритма

Простейшим элементом из которого составляется КА – это простейшая расчетная процедура (ПРП) [2]. Из ПРП составляются алгоритмические конструкции (АК) [2]. АК может иметь произвольно большой размер, так как структура ее рекурсивна. Компонентами АК могут быть как ПРП, так и АК, последние тоже могут

быть составными и т. д. Из АК составляется АКУ. Последние сочленяются иерархическим сочленением и формируют канонический алгоритм.

ПРП и АК могут быть условными или безусловными. Условные и безусловные АК и ПРП могут быть зависимыми или независимыми. АК и ПРП могут сочленяться последовательным, условным или альтернативным сочленениями.

В процессе генерации ПРП порождают операторы, АК – предложения. Операторы и предложения тоже сочленяются условным, альтернативным и последовательным сочленением. Операторы и предложения тоже могут быть условными и безусловными, а также зависимыми или независимыми.

Структурно из операторов и предложений может быть создано АТ-предложение [2]. Структура его определяется структурой А-фрагментов [2] на А-ленте [1]. Именно из АТ-предложений формируется параграф [2]. Параграфы с помощью иерархического сочленения составляют прикладной алгоритм и программу.

Состав АКУ и то, какой порядок размещения главного А-данного на входной и выходной А-лентах, – это информация достаточная для того, чтобы сгенерировать один параграф.

С точки зрения графического представления, СД – это дерево, с каждым узлом которого соотносится АКУ. СД используется как каркас или как скелет канонического алгоритма, а далее и как каркас прикладного алгоритма. Каждая АКУ порождает параграф. Алгоритм прикладной программы – это тоже дерево, узлами которого являются параграфы.

С точки зрения генерации прикладного алгоритма объектами исследования есть ПРП и их виды, способы сочленения ПРП и АК – различные вариации этих сочленений.

Описание СД в отличие от текста программы декларативно и является симбиозом текста и графики. При этом объем описания относительно меньше, чем объем текста программы.

Термин “синтез” в работе используется в сочетании с различными терминами, но обсуждается одно и то же явление. Термины зависят от контекста. Если обсуждается функциональный аспект создаваемой системы, то речь идет о синтезе процессов, иначе, если обсуждается структурный аспект, то объектом синтеза является СД. Если обсуждается внешний, изобразительный аспект – графическое изображение СД – деревья, то объектом анализа является синтез деревьев. Если обсуждается алгоритм – внутреннее представление создаваемой системы, то речь идет о синтезе алгоритмических конструкций – синтезе алгоритмов.

Алгоритмически релевантные факторы (АРФ)

В работе [1] перечислены причины, из-за которых от канонического алгоритма до реально работающей прикладной программы предстоит сделать еще ряд шагов. Эти шаги должны адаптировать алгоритм к реальным условиям обработки данных. Причины адаптации, следующие:

- алгоритмическая модель может быть сложной из-за того, что входных А-лент может быть более одной. Также и выходных А-лент может быть более одной. Источниками Р-данных могут быть базы данных, Интернет и т. п.;
- размер реальных данных, как правило, не совпадает с размерами ячеек на входных и на выходных А-лентах. Из-за этого необходимо включать в алгоритм блоки форматирования Р-данных;
- модель последовательного доступа, рассмотренная в *DS*-теории, к Р-данным самая простая. В практике электронной обработки данных есть более сложные методы доступа к данным;
- Р-данные могут поступать в режиме ONLINE от различных датчиков или в режиме диалога от оператора;
- Р-данные могут быть выведены на монитор, на различные управляющие устройства;
- Р-данные при размещении могут кодироваться, упаковываться и т. п.

В настоящее время *DS*-теория различает следующие направления работы с Р-данными порождающие группы факторов, которые усложняют канонический алгоритм на пути его преобразовании в реальный прикладной алгоритм:

- размещение Р-данных на А-ленте;
- механизм доступа к Р-данным;
- деление Р-данных;
- форматирование Р-данных;
- организация расчета;
- отражение Р-данных.

Размещение Р-данных на А-ленте. В *DS*-теории в качестве носителя Р-данных, рассматривается абстрактная А-лента. Р-данные размещаются в записях. Из записей формируются подобласти, которые могут быть вложены как компоненты в объемлющие их подобласти (рис. 2). Кроме того записи и подобласти могут объединяться и составлять А-фрагменты [2].

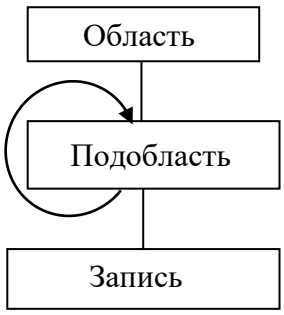

Рис. 2. Структура А-ленты

Записи и подобласти одного типа внутри объемлющей их подобласти могут быть упорядочены или нет. Записи, подобласти разных типов могут объединяться и составлять А-фрагменты. Внутри А-фрагмента записи или подобласти разных типов могут иметь порядок относительно его границ, относительно друг друга, а также иметь произвольное размещение и сочетание этих видов взаимного размещения. Такие же виды взаимного размещения возможны и между А-фрагментами. А-фрагменты могут быть как условными, так и обязательными.

В практике электронной обработки данных есть много различных видов структур хранения данных: базы данных, списки, потоки данных из локальной сети или из Интернета и т. п. Объектом исследования *DS*-теории есть виды структур хранения данных, и порядок взаимного размещения данных внутри подобных структур.

Соответствующая группа факторов определяет то, как размещаются Р-данных в подобластях на носителях-источниках.

Деление Р-данных. А-данные и С-данные, являющиеся операндами при реализации А-зависимостей, перед началом расчета могут быть разделены и размещены в различных записях или в различных подобластях. В практике электронной обработки данных это значит, что группы и совокупности данных, могут быть размещены в различных файлах. Файлы могут быть расположены как на одном носителе, так и на нескольких носителях. Соответственно, Р-данные, как абстракция реальных данных, тоже могут быть расположены на различных носителях.

Разделены могут быть А-данные и С-данные всех видов – простые, расширенные и сложные. Ситуации размещения фрагментов А-данных в областях на А-лентах возможны такие же, как и для А-данных, которые не разделены. А именно: все виды порядка размещения и все виды совместного размещения [2]. Аналогично ситуации размещения фрагментов С-данных в областях на А-лентах возможны такие же как и для неразделенных С-данных. В областях размещения фрагментов А-данных и С-данных наряду с последними могут быть Р-данные, которые являются непродуктивными для исходной СД.

Соответствующая группа факторов определяет то, сколько входных и выходных А-лент используется для хранения обрабатываемых Р-данных и то, как размещаются подобласти на этих А-лентах.

Форматирование Р-данных. В *DS*-теории внешним носителем есть А-лента. Порция хранения Р-данных, которая записывается и считывается с А-ленты, есть

запись. Размер записи не оговаривается. Предполагается, что запись по размеру совпадает с размером, сохраняемого в ней Эл-данного или простого А-данного. Из записей на А-ленте составляются подобласти. Но “запись” и “подобласть” – это не технические, а концептуальные понятия *DS*-теории. В практике электронной обработки данных обмен между компьютером и внешними устройствами выполняется блоками. Размер этих блоков или порций данных, как правило, не совпадает ни с размерами записей и подобластей, ни с размерами их содержимого – Эл-данных или простых А-данных. Блоки для хранения обмениваемой информации могут быть компонентами объемлющих их структур хранения. То есть, с технической стороны возможна иерархия порций или структур хранения информации на внешних носителях.

При выводе Р-данных на реальные носители информации записи абстрактной длины и подобласти должны быть вложены в тот формат, в котором они будут храниться на этих носителях, – должны быть форматированы. При вводе с реальных носителей информации должны быть восстановлены[4] абстрактные записи с Р-данными предназначенными для обработки.

Соответствующая группа факторов устанавливает размерные отношения между деревом типов свойств и описанием структуры подобластей с одной стороны и структурой хранения информации на реальном внешнем носителе с другой.

DS-теория учитывает то, что Р-данные могут сохраняться не только на устройствах с последовательным доступом, для измерения емкости которых, требуется одна координатная ось. Это могут быть двухкоординатные плоские экраны или различные трехкоординатные структуры.

Механизм доступа к Р-данным. При описании концептуального содержания *DS*-теории для размещения и доступа к Р-данным была использована абстрактная А-лента с записями и подобластями. Записи и подобласти – это прообраз линейных файлов с записями различных типов. Доступ к Р-данным на А-ленте простейший – последовательный. Но в практике электронной обработки данных большое разнообразие видов носителей. Соответственно, существует разнообразие способов организации данных и методов доступа к ним.

Р-данные могут размещаться в очередях различных видов. Если на устройстве доступен прямой доступ, то Р-данные, которые соотносятся с полной схемой декомпозиции, могут быть размещены в списках, что еще добавляет разнообразие в методах доступа. С помощью списков может моделироваться декомпозиция объектов принадлежащих пространствам с двумя, тремя и более размерностями.

Доступ к Р-данным может быть совмещен с одновременным кодированием или декодированием, с одновременным и архивированием или извлечением из архива. Доступ может предполагать передачу данных на большие расстояния.

К вопросам доступа с точки зрения *DS*-теории следует соотнести потребность в повторных проходах по коллекциям данных с последовательной организацией.

Факторы подобного рода составляют группу, подлежащую исследованию.

Управление расчетом. Еще одна группа факторов порождается тем, что процессами расчета необходимо управлять. Существуют различные причины и способы управления расчетами в компьютере, а именно:

- дублирование расчета на различных компьютерах;
- дублирование результирующей информации на различных носителях;
- отслеживание транзакций к базе данных выполняемых другими програм-

[4] Не существует термина, которым можно было бы обозначить процесс восстановления Р-данного – снятие или упразднение формы, в которой оно сохранялось. Наиболее близкое по смыслу понятие – деконструкция, но это из другой сферы деятельности. Понятия “распаковка”, “разархивирование”, “реформатирование” или “деформатирование” (unformat) – ассоциируются с этим процессом, но не совпадают с его сутью. В *DS*-теории используется понятие “реформатирование”.

мами. Сохранение протокола в обращении к сайту;

- прерывание и возобновление расчетов. Приостановка расчета с активацией (или без) расчета по другой программе;
- повторение расчетов с контрольных точек;
- распараллеливание расчетов;
- управление (манипулирование) окнами (мониторами);
- управление расчетами на больших расстояниях и т. п.

С точки зрения *DS*-теории в подобных ситуациях объектом декомпозиции есть сам процесс расчета. Возможность управления процессом расчета – это факторы, которые усложняют канонический алгоритм, в соответствии с которым протекает расчет. Помимо признаков, которые описывают ситуацию управления, существуют Р-данные, которые необходимы для управления процессом расчета.

Отражение на мониторе. А-лента – это носитель, измерение которого производится с помощью одной координатой оси, но для размещения информации используются различные табло, экраны и т. п. – для измерения поверхности изображения, которых, используются две координатные оси. В связи с выводом Р-данных на такие носители (и в связи с вводом с подобных носителей) определяется еще одна группа факторов. Они описывают размещение Р-данных в окнах, порядок вывода Р-данных в окно. Учитывая то, что на экран монитора могут быть выведены текст, графика – векторная и растровая, – видео, группа этих факторов обширна и влияние ее на канонический алгоритм весьма существенное.

В случае отражения на мониторах выводимые (и вводимые) Р-данные сами по себе могут быть объектами декомпозиции и поэтому надо исследовать не просто влияние факторов отражения Р-данных на канонический алгоритм, а синтез схем декомпозиции Р-данных.

Влияние АРФ на канонический алгоритм. В процессе адаптации канонического алгоритма может возникнуть необходимость учитывать факторы любой из вышеперечисленных групп. Группы, факторы, влияющие на канонический алгоритм, могут влиять каждая отдельно или в любом сочетании – даже все вместе. Перечень групп АРФ не завершен и вполне может быть дополнен новыми группами.

Особенные свойства *DS*-теории

Практически общепризнано, что программная инженерия нуждается в научной теории [7 – 12]. Далее приведены некоторые особенности *DS*-теории как научной теории.

Нововведения. В *DS*-теории вводится ряд понятий, позволяющих глубже понимать природу алгоритма и программы. Абстракция – одно из важных понятийных средств в программировании. Но исключительно важны *информативно емкие* абстракции, емкие понятия [13 – 16]. В *DS*-теории такими понятиями являются совокупное свойство, алгоритмическая зависимость, квалификационное предложение [1], СД и КС.

Понятие СД позволило разделить функциональность и управление алгоритма и программы. Введены понятия функционального ядра, алгоритмического фрейма, функционального содержания и алгоритмической матрицы [2].

Вместо понятия “данное” введены и используются понятия “Р-свойство” [1], “размещение”, “Р-данное” и “Д-данное” [2].

Введено понятие порядка при размещении Р-данных на внешних носителях. Р-данные могут быть перемешаны в областях и подобластях на носителях. То, каким образом они перемешаны – характер смешения, – определяется и описывается формально.

Разделены DPS и дерево алгоритма. Каждое из этих понятий может быть изображено деревом, но назначение этих описаний различное[5].

В *DS*-теории введено понятие “Квалификационного предложения”. Эта идея создает предпосылки манипулирова-

[5] Это был вполне ожидаемый шаг в развитии R-технологии [17] и JSP [18].

ния квалификационными предложениями. Подобно тому, как есть операции над схемами декомпозиции, так возможны операции над квалификационными предложениями. С этой точки зрения *DS*-теория есть альтернативой реляционной модели данных [19].

В *DS*-теории созданы условия для частичной классификации алгоритмов[6]. Группы АРФ определяют группы алгоритмов реализующие их. В *DS*-теории проводится исследование групп алгоритмов вместо проектирования универсальных классов или универсальных алгоритмов. Вместо того, чтобы исследовать многообразие алгоритмов, исследуются факторы[7], которые влияют на алгоритмы.

Разделение поля алгоритмов на группы позволяет оценить уровень издержек в каждой из групп и после этого уделить особое внимание исследованию групп критичных с точки зрения издержек[8]. Исследование каждой из групп АРФ сужает поле алгоритмов с точки зрения неопределенности и неуправляемости. Результат исследования внутри группы – это рекомендации по генерации алгоритмов внутри этой группы, это общие принципы генерации.

В *DS*-теории важным есть прозрачный механизм синтеза алгоритмов и содержимое библиотеки алгоритмических примитивов. Создание подобной библиотеки имеет аналог в машиностроении – это унификация узлов и деталей. Как для машиностроения унификация – это средство сокращения издержек, так для программирования создание библиотеки унифицированных примитивов – это средство сокращения издержек в построении алгоритмов. Хотя *DS*-теория создает предпосылки автоматической генерации алгоритмов, а не ручного проектирования. Механизм синтеза, кроме того, что он дополняемый и расширяемый, также открытый для тестирования подобно тому, как открыты для анализа доказательства теорем в математике.

Дополнительные ожидаемые результаты. Как упоминалось в [2], несмотря на то, что алгоритм как результат генерации, представляется с помощью алгоритмического языка императивного типа, СД, как описание исходное для генерации, является описанием декларативного типа. Сгенерированный алгоритм по определению отражает порядок вычислений, а описание СД явно этот порядок не показывает. В процессе генерации алгоритма учитывается порядок обхода DPS сверху вниз и слева направо и благодаря этому он появляется в сгенерированном алгоритме. По сути, СД является объектом декларативного программирования. *DS*-теория благодаря системе емких понятий позволяет создать систему умолчаний в программировании. Можно говорить об инкапсуляции алгоритма, об умолчании описания алгоритма. Но пока можно говорить только о последовательном приближении к такому стилю программирования по мере исследования видов СД и групп АРФ.

Декларативный стиль программирования обеспечивает описание статического объекта, при этом алгоритм – это описание динамического объекта. Последнее сложнее по сравнению с описанием статического объекта. Но и в этом отношении можно говорить только как о постепенном и последовательном приближении от одного вида описания к другому. Удельный вес статического описания будет увеличиваться, а динамического описания уменьшаться.

---

[6] В науке существует прецеденты подобной классификации. Например, идея “атомного веса” создала предпосылки к классификации химических элементов.

[7] Нечто подобное имело место в истории математики. Математики Абель и Галуа не стали искать универсальное решение алгебраических уравнений пятой, шестой и больших степеней, а исследовали свойства коэффициентов, от которых зависит решение. Это дало возможность ответить на вопрос о возможности нахождения корней в принципе. Такой же подход реализован при исследовании алгоритмов.

[8] Подобно тому, как для создания вакуума в замкнутом пространстве различными химическими реактивами связывают различные газы – компоненты воздуха, пока не достигают необходимого уровня, так исследуя фактор за фактором, от поля алгоритмов постепенно отсекают группы алгоритмов. Как один реактив связывает один газ, подобно этому один фактор выделяет одну группу из поля алгоритмов.

DS-теория ориентирована на создание (генерацию) уникального специализированного кода и объективно необходимого количества подпрограмм. А специализированный код – это более плотный и продуктивный код.

Математические аспекты *DS*-теории. В *DS*-теории предпринимаются попытки к тому, чтобы рассматриваемым явлениям, объектам и зависимостям давать в плане точности определения максимально приближенные к математическим определениям. Здесь есть все предпосылки к тому, чтобы *DS*-теория как теория прикладных алгоритмов стала частью математической науки подобно классической теории алгоритмов[9] [20].

В *DS*-теории прилагаются усилия к тому, чтобы любую изучаемую или порождаемую конструкцию понимать (или рассматривать) как результат операций над некоторыми примитивами. Таких видов конструкций есть несколько:

- любая СД – это либо примитив (ЧСД), либо результат операций над ЧСД;
- АК или АКУ – это, как правило, результат операций над ПСД [2]. Операциями есть последовательное, условное, альтернативное сочленения;
- АКУ являются операндом, к которому применяется иерархическое сочленение при построении дерева полной схемы декомпозиции;
- параграфы – это результат операций над операторами и предложениями. Операции – последовательные, условные, альтернативные сочленения.

Эти усилия формализации направлены на то, чтобы: а) конструкции, составляемые человеком, могли быть подвергнуты семантическому контролю – как предварительному шагу по устранению ошибок; б) обеспечить логический вывод алгоритмов.

Алгоритмические конструкции в реальных алгоритмах могут быть весьма сложные. Но любая подобная конструкция – это только очередной шаг в растущей сложности алгоритмов. В предшествующих шагах конструкции были проще. Последующие шаги в проектировании алгоритмов вполне могут быть отмечены более сложными алгоритмами. Усилия в *DS*-теории прилагаются с целью найти причины растущей сложности. А для этого требуется формализация и математизация процесса анализа алгоритмов.

Механизм контроля результатов. Контроль результатов, которые обеспечивает *DS*-теория, имеет два аспекта – результаты, получаемые в процессе исследований и результаты, получаемые в процессе генерации конкретной прикладной программы.

Результат исследований – это система алгоритмических примитивов и механизм синтеза алгоритмов. Так как теория имеет свою внутреннюю логику – развивается от единой парадигмы, выше объявленной, то контролироваться должен процесс построения системы примитивов и сама система, а также механизм синтеза. Это разовая процедура и она может быть выполнена только умозрительно (с карандашом в руках). Но эти объекты обозримы (в отличии от текстов многих реальных программ) и вполне могут быть доступны многим исследователям подобно тому как доступны для изучения и контроля доказательства теорем в математике.

Конкретная программа генерируется на основании описания конкретной СД и набора связанных с ней АРФ. На этом этапе возможен произвол того, кто составляет эти исходные данные и, соответственно, возможны ошибки. Здесь предусматривается проверка КС. Если КС замкнут, то это значит, что алгоритм программы корректен. Как упоминалось выше, для проверки КС исходные конструкции подвергаются семантическому контролю. Но следует понимать, если формулы или отношения между исходными данными – функциональность СД – некорректны по сути, то средствами *DS*-теории это определить невозможно.

[9] Исследование алгоритмов и групп алгоритмов в *DS*-теории проводится так же, как проводились (и проводятся) исследования алгоритмов в классической теории алгоритмов. Но при этом цели исследований различные. Для *DS*-теории приоритетная цель – синтез алгоритмов и работа по достижению этой цели выполняется системно.

И, самое главное, основной критерий корректности программы – это корректный результат расчета. *DS*-теория ориентирована на последовательное сужение поля ошибок, но средствами для полного их устранения не располагает.

Возможность развития. Исходя из текущего состояния *DS*-теории, рассматриваются следующие направления развития.

В настоящее время в различных сферах человеческой деятельности исследуются и проектируются объекты многомерные, динамичные, многофункциональные и многофакторные. Объекты могут находиться в микро или в макромире, в труднодоступных средах. Все это может быть источником разнообразия СД с точки зрения КА. Поэтому исследуются виды и компоненты СД и операции над ними. Особенно важны операции синтеза СД.

Выше в статье приведен перечень АРФ. В процессе поисковых исследований этот перечень может быть дополнен. *DS*-теория благодаря идее АРФ позволила разделить поле алгоритмов на секторы. Внутри каждого сектора очерчены (выделены) группы простых АК и механизмы синтеза реальных прикладных алгоритмов. Эти обстоятельства определяют одно из направлений исследований – поиск новых групп АРФ и анализ алгоритмов связанных с этими группами.

Объекты, подлежащие декомпозиции – это, прежде всего, процессы. Притом, это процессы, которые создаются, проектируются. Синтез процессов (проектирование процессов) – это наиболее важное и наименее развитое направление *DS*-теории.

С точки зрения конкретных прикладных программных комплексов *DS*-теория в настоящее время – это теория достаточно общего плана, хотя она уже применима на более конкретном практическом уровне. Существенный эффект может быть получен от *DS*-теории, когда будут разработаны специализированные схемы декомпозиции. Это в области АСУ [21], распределенной обработки данных – в тех областях, где большие масштабы программирования электронной обработки данных.

Еще одно из направлений исследования – это целенаправленный поиск изобразительных средств – двумерная графика. Но эффективная графика может быть получена только в том случае, если будет система эффективных емких понятий.

Сравнение программирования с машиностроением

DS-теория создает предпосылки для создания некоторого другого вида производства, по сравнению с тем, что реально существует в производстве программ в настоящее время. Речь идет о предпосылках создания индустриального производства программ. Можно проследить некоторые аналогии.

Разделение труда. В проектировании архитектуры программных систем, в проектировании алгоритмов и программировании нет глубокого объективного разделения труда. В машиностроении разделяется описание конструкции изделия и технологии производства. Соответственно, есть разделение на конструирование, технологическую подготовку производства и само производство. Сейчас программирование – это конструирование и изготовление одновременно. Должно быть:

- конструирование;
- технологическая подготовка производства;
- инструментальная подготовка производства;
- изготовление.

В машиностроении есть более глубокая детализация, но в программной инженерии желательно дойти хотя бы до этого уровня. Желаемого уровня разделения труда нет по той причине, что текст программы – это уже конечный продукт, а описания изделия не существует. Имеется в виду компактное и желательно двумерное описание изделия и притом детальное. Описание схемы декомпозиции – это описание конструкции, текст программы – конечное изделие.

Проблема программной инженерии заключается в том, что текст программы,

написанный с использованием традиционных строчных языках очень громоздкий и труднообозримый. В *DS*-теории предпринимается попытка решить эту проблему. То описание СД, которое приведено в этой и предыдущих статьях еще далеко от желаемого. Такое положение обусловлено тем, что в описании надо совместить сжатую компактную запись и детализацию[10]. Но предпосылки для создания компактного описания – это система емких понятий, предлагающиеся *DS*-теорией.

DS-теория предлагает и использует несколько схем деления алгоритмов. Разделяются дерево полной схемы декомпозиции и дерево алгоритма. Полная схема декомпозиции разделяется на АКУ. Разделяются дерево типов свойств и описание структуры областей размещения. Слои АРФ это еще одно направление деления алгоритмов [1]. Разделяются алгоритмическая матрица и функциональное наполнение. Следует также упомянуть тот факт, что операции над СД выполняются задолго до того, как начинается описание операционной среды, в которой будет реализована схема декомпозиции. Эти схемы деления создают объективные предпосылки к разделению труда в программировании.

Разработка и доработка изобразительных средств и генераторов алгоритмов должны быть подобны работе инструментального производства в машиностроении. В машиностроении постоянно разрабатывается оснастка, внедряются новые технологии с целыми техническими комплексами, что реализуют новые технологии. Подобно этому в программной индустрии постоянными должны быть доработка генераторов алгоритмов и изобразительных средств.

Целенаправленное и системное сокращение издержек. *DS*-теория создает условия для того, чтобы проводить исследования и разрабатывать (или дорабатывать) инструмент и среду проектирования, и не ждать когда появится очередной метод или технология [10, 11]. В машиностроении действует другой принцип – здесь не ожидается появление новых методов, но для трудоемких операций или переделов целенаправленно разрабатываются технологии, инструмент, оснастка. Если технологическая операция, технологический передел или технология в целом влекут большие издержки, то инженеры и исследователи планомерно ищут новые технологические средства, чтобы сократить эти издержки.

Именно для этого в *DS*-теории производится классификация. Предполагается, что сокращение издержек эффективно внутри групп алгоритмов, порождаемых АРФ, а также внутри групп вариаций канонического алгоритма порождаемых видам систем.

Конвейер или поточная линия [22] – это наиболее мощное средство экономии времени и издержек изобретенных в машиностроении и таким остается и в настоящее время. В программировании поточная линия в традиционном понимании невозможна. Но это не значит, что в программировании не может быть тотальной системной экономии издержек. Емкие понятия, система умолчаний и генерация алгоритмов – вот те ключевые факторы, что позволят целенаправленно и планомерно сокращать издержки в программировании.

Гносеологический аспект *DS*-теории. В промышленности протекают, активно взаимодействуя, два процесса: производство (изготовление продукции) – практика, и научные исследования – теория. Из производства наука пополняется фактами, задачами и проблемами. Теория обобщает, индуцирует факты, явления, а

[10] Первый и необходимый шаг в повышении производительности благодаря разделению труда в промышленности был сделан в арсеналах вооружений Англии, Германии, Франции, Швеции в XIX веке. Именно там были детальнейшим образом описаны и изображены на чертежах комплектующие компоненты изготавливаемого оружия. Именно полный и точный чертеж детали позволил записать технологию ее изготовления и тем самым создать условия к разделению труда. Чертеж детали обеспечил отчуждение производственной операции от субъективного мастерства человека и обеспечил дальнейшую специализацию производственных операций. Чертеж детали позволил моделировать как конструкцию, так и технологию, что позволило существенно сократить издержки изготовления.

затем дедуктивно поставляет в производство решения задач и проблем, это, как правило, новые технологии. Благодаря такой деятельности поддерживает активность науки. Взаимодействие этих двух процессов – источник постоянного развития промышленности. В индустрии программного обеспечения есть только один из упомянутых процессов – производство. Науки нет и, следовательно, практически нет развития.

Взаимодействие науки и производства далеко не безоблачно – возможны и сбои. Если новую задачу в науке не получилось разрешить, то поиск вполне продолжается и дальше. Теория может развиваться самостоятельно – не обязательно по запросу практики. Кроме того, могут быть ложные выводы, результаты могут получаться слабые или бесполезные (как может оказаться впоследствии). Это вынужденные издержки в науке. Но они не останавливают ее развитие. Несмотря на то, что негативные явления возможны и в отношении *DS*-теории, она должна развиваться.

Разработка *DS*-теории – это попытка создать параллельно текущий процесс научного развития для процесса производства программ. Сомнения может вызывать обобщенная модель *DS*-теории – СД. Можно сомневаться в том, насколько адекватны модели и алгоритмы прикладных программ, выведенных из обобщенной СД, какой уровень автоматизации или генерации алгоритмов, насколько алгоритмы адекватны спецификациям. Можно сомневаться в том, насколько *DS*-теория логически целостна и самодостаточна, но то, что научный процесс должен дополнить программную инженерию не должно вызвать сомнения.

Набор новых емких понятий, которые предложены *DS*-теорией, не предполагается завершенным. Наряду с тем как выше предложено организовать целенаправленный процесс сокращения издержек, так предлагается организовать целенаправленный и сознательный поиск и разработку емких понятий. Предпосылки для этого существуют.

Выводы

DS-теория представлена как научная теория. Описаны ее атрибуты – парадигма, поле исследований, цель исследований, основная задача, метод решения, основная теоретическая модель – СД. Описаны виды СД и операции на СД. Показано что СД есть описание, исходное для генерации алгоритмов. Также показано, что СД как описание заменяет описание алгоритма и при этом остается декларативным в противоположность описанию алгоритма, как императивному. Описаны АРФ, которые следует учитывать при генерации алгоритмов для того, чтобы алгоритмы становились реальными. В работе акцентируется внимание на формализации и математическому описанию явлений и объектов *DS*-теории. Предложен механизм контроля выводов и результатов теории.

В работе описано направление развития *DS*-теории. Приоритетное направление соотносится с созданием и развитием системы емких понятий и абстракций. С точки зрения практического применения *DS*-теории предложен подход подобный тому, что есть в машиностроении – развитие и продвижение передовых технологий по запросу, по потребности – как результат целенаправленных исследований. Новые методы программирования и синтеза алгоритмов должны быть результатом целенаправленных исследований.

1. *Колесник В.Г. DS*-теория как прототип теории прикладных алгоритмов. Проблеми програмування. 2012. № 1. С. 17–33.
2. *Колесник В.Г. DS*-теория. Представление канонического алгоритма с помощью алгоритмического языка. Проблеми програмування. 2015. № 1. С. 3–18.
3. *Колесник В.Г. DS*-теория. Исследование факторов деления Р-данных с целью генерации прикладных алгоритмов. Первая часть. Проблеми програмування. 2015. № 3. С. 3–12.
4. *Колесник В.Г. DS*-теория. Исследование факторов деления Р-данных с целью генерации прикладных алгоритмов. Вторая часть. Проблеми програмування. 2015. № 4. С. 3–13.

5. *Колесник В.Г. DS*-теория. Исследование факторов форматирования Р-данных. Проблеми програмування. 2016. № 4. С. 14–26.
6. *Бертран Мейер.* Объектно-ориентированное конструирование программных систем, 2-е издание. Русская редакция. 2005. 1204 с.
7. *Vincent Rosener and Denis Avrilionis.* Elements for the Definition of a Model of Software Engineering. OneTree Technologies. [Electronic resource]. URL: http://www.irisa.fr/lande/lande/icse-proceedings/ gamma/p29.pdf
8. More comments on SEMAT. [Electronic resource] // URL: http://blog.hexacta.com/more-comments-on-semat/
9. *Dewayne E. Perry and Don Batory*. On the Structure of General Theories of Software Engineering. – The University of Texas at Austin. [Electronic resource]. URL: http://users.ece.utexas.edu/~perry/work/papers/gtse-structure.pdf
10. *Ivar Jacobson and Ed Seidewitz.* What happened to the promise of rigorous, disciplined, professional practices for software development? Ivar Jacobson International. Communications of the ACM, Vol. 57, N 12. P. 49–5410.
11. *Victor R. Basili.* The role of experimentation in software engineering: past, current, and future. Proceedings of the Eighteenth International Conference on Software Engineering (ICSE), Berlin, Germany, March 1996.
12. *National* Science Foundation. Computing and Communication Foundations (CCF). The Algorithmic Foundations (AF) program. [Electronic resource] // URL: http://www.nsf.gov/cise/ccf/af_pgm15.jsp http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503299&org=CCF&from=home
13. *Dijkstra E.W.* On the role of scientific thought (EWD447). [Electronic resource] // URL: http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html
14. *Victor R. Basili.* Learning through Application. The Maturing of the QIP in the SEL – [Electronic resource] // URL: https://www.cs.umd.edu/~basili/publications/chapters/C29.pdf.
15. *Brooks F.P.* The Mythical Man Month – Silver Anniversary Edition, Addison-Wesley, 1995.
16. *Edmund M. Clarke, Jeannette M. Wing, Et Al.* Formal Methods: State of the Art and Future Directions. ACM Computing Surveys. 1996. Vol. 28, N 28. P. 626–643.
17. *Вельбицкий И.В.* Технология программирования. Киев: Техніка, 1984. 250 с.
18. *Jackson M.A.* Principles of Program Design. New York: Academic Press, 1975.
19. *Дж. Дейт.* Введение в системы баз данных. Восьмое издание. К.; 1328 с. Вильямс, 2008, 3 кв.
20. *Алферова З.А*. Теория алгоритмов. М.: Статистика. 1973. 164 с.
21. *Колесник В.Г., Семченко В.А.* Концептуальное проектирование сети ЭВМ для цеховой АСУ. Донецк: ИЭП АН УССР, 1989. 16 с.
22. *Генри Форд.* Моя жизнь, мои достижения. 5-е изд. Попурри, 2014. 352 с.

References

1. Kolesnyk V.G. DS-theory as a prototype of the theory of applied algorithms. Problems in programming. 2012. N 1. P. 17–33. (In Russian).
2. Kolesnyk V.G. DS-theory. Presentation of canonical algorithm by means of algorithmic language. Problems in programming. 2015. N 1. P. 3–18. (In Russian).
3. Kolesnyk V.G. DS-theory. Research of R-data division factors in order to generate applied algorithms. Part 1. Problems in programming. 2015. N 3. P. 3–12. (In Russian).
4. Kolesnyk V.G. DS-theory. Research of R-data division factors in order to generate applied algorithms. Part 2. Problems in programming. 2015. N 4. P. 3–13. (In Russian).
5. Kolesnyk V.G. DS-theory. The research of R-data factors formatting. Problems in programming. 2016. N 4. P. 14–26. (In Russian).
6. Bertrand Meyer. Object-Oriented Software Construction, 2nd Edition. Russian Edition, 2005. 1204 p.
7. Vincent Rosener and Denis Avrilionis. Elements for the Definition of a Model of Software Engineering. OneTree Technologies. [Electronic resource]. URL: http://www.irisa.fr/lande/lande/icse-proceedings/ gamma/p29.pdf
8. More comments on SEMAT. [Electronic resource] URL: http://blog.hexacta. com/more-comments-on-semat/

9. Dewayne E. Perry and Don Batory. On the Structure of General Theories of Software Engineering. – The University of Texas at Austin. [Electronic resource] URL: http://users.ece.utexas.edu/~perry/work/papers/gtse-structure.pdf
10. Ivar Jacobson and Ed Seidewitz. What happened to the promise of rigorous, disciplined, professional practices for software development? Ivar Jacobson International. Communications of the ACM, Vol. 57, N 12. P. 49–5410.1145/2677034.
11. Victor R. Basili. The role of experimentation in software engineering: past, current, and future. Proceedings of the Eighteenth International Conference on Software Engineering (ICSE), Berlin, Germany, March 1996.
12. National Science Foundation. Computing and Communication Foundations (CCF). The Algorithmic Foundations (AF) program. [Electronic resource] URL:
http://www.nsf.gov/cise/ccf/af_pgm15.jsp
http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503299&org=CCF&from=home
13. Dijkstra E.W. On the role of scientific thought (EWD447). [Electronic resource] URL: http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html
14. Victor R. Basili. Learning through Application. The Maturing of the QIP in the SEL – [Electronic resource] URL:
https://www.cs.umd.edu/~basili/publications/chapters/C29.pdf
15. Brooks F.P. The Mythical Man Month – Silver Anniversary Edition, Addison-Wesley, 1995.
16. Edmund M. Clarke, Jeannette M. Wing, Et Al. Formal Methods: State of the Art and Future Directions. ACM Computing Surveys. 1996. V. 28, N 28. P. 626–643.
17. Velbitsky I.V. "Technology of programming." Tekhnika, Kiev, 1984. (In Russian).
18. Jackson M.A. Principles of Program Design. New York: Academic Press, 1975.
19. C.J. Date. An Introduction to Database Systems, 8th Edition ruction. Williams; 2008. К.; 1328 p.
20. Alferova Z.A. The theory of algorithms. Moscow: Statistics, 1973. 164 p. (In Russian).
21. Kolesnyk V.G. Semchenko V.A. Conceptual design of computer network for shop floor control system. Doneck, 1989. 16 p. (In Russian).
22. Henry Ford. My Life and Work, Doubleday, Page & company; 1922. 140 p.

Получено 04.01.2017

Об авторе:

Колесник Валерий Георгиевич,
старший научный сотрудник
кафедры АПП.
Количество научных публикаций в
украинских изданиях – 25.
http://orcid.org/0000-0002-2313-9852.

Место работы автора:

Донбасская государственная
машиностроительная академия.
г. Краматорск, ул. Шкадинова, 72,
п/я 13.