КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ
И МОДЕЛИРОВАНИЕ 2014 Т. 6 № 2 С. 331–344

Ки&М

ПРИКЛАДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ И ИНФОРМАЦИОННЫЕ СИСТЕМЫ

УДК: 004.02

Методика работы с унаследованными информационными системами

Н. С. Калуцкий

ООО «Прогресстех-Дубна»,
Россия, 141980, Московская обл., г. Дубна, ул. Программистов, д. 4

E-mail: nikita.kalutsky@progresstech.ru

Получено 15 марта 2014 г.

В данной статье предлагается методика работы с унаследованными информационными системами. В процессе профессиональной деятельности специалистам в области машиностроения приходится сталкиваться с тем, что компьютерные приложения, с помощью которых было спроектировано изделие, устаревают значительно быстрее, чем само изделие. В тоже время переход на современные информационные системы может быть затруднен по ряду причин. В таком случае говорят о проблеме унаследованных систем. Она возникает тогда, когда жизненный цикл изделия намного превосходит время жизни программных систем, с помощью которых это изделие создавалось. Для решения этой проблемы в настоящей статье автором предлагается методика, на основе которой был разработан программный комплекс.

Ключевые слова: унаследованная система, информационная модель, программный комплекс

Methodic of legacy information systems handling

N. S. Kalutsky

LLC "Progresstech-Dubna", 4 Programmistov st., Moscow reg., Dubna, 141980, Russia

Abstract. — In this article a method of legacy information systems handling is offered. During professional activities of specialists of various domains of industry they face with the problem that computer software that was involved in product development stage becomes obsolete much quickly than the product itself. At the same time switch to any modern software might be not possible due to various reasons. This problem is known as "legacy system" problem. It appears when product lifecycle is sufficiently longer than that of software systems that were used for product creation. In this article author offers an approach for solving this problem along with computer application based on this approach.

Keywords: legacy system, data model, software application

Citation: *Computer Research and Modeling*, 2014, vol. 6, no. 2, pp. 331–344 (Russian).

Введение

В связи с бурным развитием информационных технологий за последние два десятилетия многие программные продукты, бывшие актуальными ещё несколько лет назад, сегодня признаются устаревшими. В то же время легко увидеть, что многие изделия машиностроения, разработанные десятки лет назад, до сих пор находятся в серийном производстве и широкой эксплуатации. Рассмотрим описанное явление на примере авиастроения.

- Разработка модели Боинг 747 (первый полет в 1968 году) была выполнена более 40 лет назад. Модифицированный вариант Боинг 747-8 выпускается и в настоящее время.
- Разработка бомбардировщика Б-52 была произведена в 1950-х годах (на вооружении ВВС США с 1955 года). На сегодня по-прежнему остаётся основным самолётом дальней бомбардировочной авиации американских ВВС и будет оставаться таковым до 2030 года.
- Одна из наиболее популярных моделей для пассажирских авиаперевозок на сегодняшний день — Эрбас А320, была разработана в 1980-х.

Очевидно, что вычислительная техника даже конца 1980-х годов не могла предоставить таких богатых вычислительных возможностей, какие есть сегодня. В связи с этим сегодня наблюдается порой парадоксальная ситуация, когда крупные и коммерчески успешные предприятия продолжают использовать сильно устаревшие программное и аппаратное обеспечение по причине высокой сложности переноса накопленной за годы информации на современные программные и аппаратные платформы. Рассмотрению этой проблемы и посвящена настоящая статья.

Обзор нормативно-законодательной базы

На сегодняшний день все основные процессы, связанные с поддержкой жизненного цикла программных средств, определены в стандарте ГОСТ Р ИСО/МЭК 12207 [5]. В нём определена общая структура процессов жизненного цикла программных средств, на которую нужно ориентироваться в сфере информационных технологий. Стандарт охватывает процессы, виды деятельности и задачи, которые используются, начиная с этапа приобретения программного продукта и заканчивая сопровождением и прекращением применения программного продукта. Для рассматриваемого в статье этапа жизненного цикла в стандарте определен вид деятельности *перемещение* внутри процесса *сопровождения программных средств*. При этом должен быть разработан, документирован и выполнен план перемещения. Запланированные действия должны включать в себя участие пользователей. План должен содержать следующие разделы:

a) анализ требований и определение перемещения;
b) разработку инструментария перемещения;
c) конверсию программного продукта и данных;
d) выполнение перемещения;
e) верификацию перемещения;
f) поддержку прежней среды в будущем.

Кроме того, стандарт [ГОСТ Р ИСО/МЭК 12207-2012, 2012] выделяет отдельный процесс *прекращения применения программных средств*.

Добавим, что процесс сопровождения программных средств, представленный в стандарте, является частным случаем более широкого понятия процесса технического обслуживания систем, приведенного в [ГОСТРИСО/МЭК 15288-2005, 2006].

Обзор существующих методов

На сегодня в мировой практике сформировалось множество различных подходов к решению проблемы унаследованных систем. Упрощая, можно свести множество подходов к следующему набору:

1) замена графического интерфейса пользователя (ГИП);
2) эмуляция;
3) виртуализация;
4) снятие ПО с эксплуатации.

Вкратце, остановимся на каждом из перечисленных методов.

Замена ГИП. В случае если невозможно полностью заменить унаследованную систему на современный аналог, всё же существуют некоторые возможности по повышению эффективности работы с нею. Сегодня многие разработчики идут по пути замены графического интерфейса пользователя. Так, большое распространение получила практика добавления веб-интерфейса к старым терминальным консольным приложениям (terminal-based mainframe application) [McCormick, 2000]. Недостатки такого подхода в следующем:

- снижение производительности системы из-за увеличенного времени прохождения запросов;
- снижение производительности оператора системы из-за работы с манипулятором типа «мышь».

Достоинства подхода:

- веб-интерфейс прост в использовании и не требует специальных знаний;
- допускается работа неквалифицированных пользователей.

В литературе данный подход получил название middleware или промежуточное приложение [Middleware…, 2013].

Эмуляция. Термином эмуляция (англ. emulation) обозначают явление воспроизведения программными или аппаратными средствами либо их комбинацией работы других программ или устройств. Выделяют следующие разновидности эмуляции:

- аппаратная;
- программная;
- программно-аппаратная эмуляция.

Эмуляция позволяет выполнять компьютерную программу на платформе (компьютерной архитектуре и/или операционной системе), отличной или в некоторых случаях идентичной той, для которой она была написана в оригинале. Эмуляцией также называют сам процесс этого выполнения. В отличие от симуляции, которая лишь воспроизводит поведение программы, при эмуляции ставится цель точного моделирования состояния имитируемой системы для выполнения оригинального машинного кода.

При использовании языков высокого уровня, иногда в целях сохранения быстродействия исполняемой программы, вместо эмуляции делают портирование программ в новую среду. В этом случае производится переписывание заново аппаратно-зависимых участков кода.

Теоретически, согласно тезису Чёрча–Тьюринга [Тьюринг, 2003], любая операционная среда может быть эмулирована в любой другой среде. На практике, однако, встречается ряд трудностей; в частности, точное поведение эмулируемой системы часто недокументированно (или скрывается под грифом коммерческой тайны) и должно быть исследовано и определено с помощью обратной разработки.

Достаточно полная эмуляция некоторой аппаратной платформы требует предельной точности, до уровня отдельных тактовых циклов, недокументированных особенностей и даже ошибок реализации. В противоположность этому, на некоторых других платформах мало использовался прямой доступ к оборудованию. В этом случае оказывается достаточным обеспечить некоторый уровень совместимости, обеспечивающий трансляцию системных вызовов эмулируемой системы в вызовы работающей системы.

Виртуализация. Виртуализация в вычислениях — процесс представления набора вычислительных ресурсов или их логического объединения, который даёт какие-либо преимущества перед оригинальной конфигурацией. Это новый подход к использованию вычислительных ресурсов, неограниченных реализацией, физической конфигурацией или географическим положением. Под вычислительными ресурсами будем понимать вычислительные мощности и хранилища данных. Другими словами виртуализация — это изоляция вычислительных процессов и ресурсов друг от друга.

Типы виртуализации приводятся ниже.

1) Программная виртуализация:
   - динамическая трансляция;
   - паравиртуализация;
   - встроенная виртуализация.
2) Аппаратная виртуализация.
3) Виртуализация на уровне операционной системы.

Снятие ПО с эксплуатации. Снятие с эксплуатации (англ. application retirement) это практика вывода из производственной эксплуатации устаревшего программного обеспечения. Также снятие с эксплуатации может применяться к такому ПО, чьи функциональные возможности дублируются другими имеющимися у предприятия ПО.

При снятии с эксплуатации необходимо обеспечить доступ к данным приложения, накопленным за весь, зачастую довольно продолжительный, срок его службы. Данные приложения должны быть надежно сохранены и на последующий длительный период времени. Эти требования предъявляется законодательствами многих стран в части обработки и хранения информации. Поэтому унаследованная система не может быть просто остановлена и отключена от питания — вся содержащаяся в ней информация должна быть классифицирована, для неё должен быть определен срок хранения и режим доступа к ней на случай аудита или судебного разбирательства.

Унаследованные системы зачастую поддерживаются в работоспособном состоянии с единственной целью предоставления доступа к унаследованным данным в исключительных случаях. При этом известно, что в структуре затрат предприятия на ПО до 75 % средств тратится только на текущую поддержку работоспособности имеющегося ПО [Murphy, 2006]. Отсюда следует, что своевременное списание унаследованных систем дает существенный экономический эффект, достигаемый за счет сокращении затрат на обслуживание, администрирование и лицензирование программного и аппаратного обеспечения. По данным, приводящимся в [Heine, 2006], экономия до 20 % от базовой стоимости ПО может быть достигнута при строгом следовании стратегии регулярной ревизии и вывода из эксплуатации имеющегося у предприятия ПО. Однако определение того, стоит ли идти на снятие ПО с эксплуатации или нет, является нетривиальной задачей.

Описание предлагаемого метода

В настоящей работе предлагается модифицированный метод снятия устаревшего ПО с эксплуатации через миграцию его данных на современное программное и аппаратное обеспечение. Весь процесс можно разбить на следующие этапы:

- перенос или миграция данных;
- выбор и внедрение инструмента доступа к данным.

Рассмотрим эти этапы подробнее.

Миграция данных. Миграция данных — это процесс переноса данных из одной системы хранения в другую, отличающуюся от первой:

- типами хранилища;
- форматами представления данных;
- системам управления данными (СУБД).

Миграция данных проводится в случаях замены используемой системы управления данными, а также в случае слияния данных из разных источников (хранилищ).

Для обеспечения процедуры миграции данные из исходной системы проецируются в целевую систему на основе принятой схемы трансформации данных. Схема трансформации представляет собой набор правил по переводу данных из формата исходной системы в данные формата целевой системы с соблюдением требований по представлению данных в целевой системе.

Процедура миграции данных может состоять из многих этапов, но выделяют три главных этапа:

- экстракция данных (этап считывания данных из исходной системы);
- трансформация данных (этап обработки и изменения данных в соответствии с требованиями целевой системы);
- загрузка данных (этап записи данных в целевую систему).

В международной практике описанный процесс получил название ETL-process (extract, transform and load).

Часто к упомянутым этапам добавляют этап проверки подлинности (валидации) данных. На этапе валидации данные проходят проверку на предмет соответствия их требованиям целевой среды.

После загрузки данных в целевую систему она должна быть проверена (процесс верификации) на предмет целостности и верности данных для того чтобы убедиться, что перенос данных прошел успешно.

Часто во время процесса миграции производится ручная или автоматизированная очистка данных, т. е. удаление излишней или устаревшей информации, что благотворно сказывается на качестве данных. В свою очередь трансформирование данных под требования по представлению данных в целевой, обычно более совершенной, системе, также может повышать качество данных.

На практике для достаточно сложных приложений промышленного класса описанные выше фазы миграции данных (проектирование, экстракция, очистка, загрузка, верификация) должны быть повторены несколько раз, прежде чем новая система может быть развернута для производственной эксплуатации.

Существует 4 основных вида миграции данных:

1) миграция на новое физическое хранилище данных;
2) миграция на новую СУБД;
3) миграция на новое программное обеспечение;
4) миграция на новые бизнес-процессы.

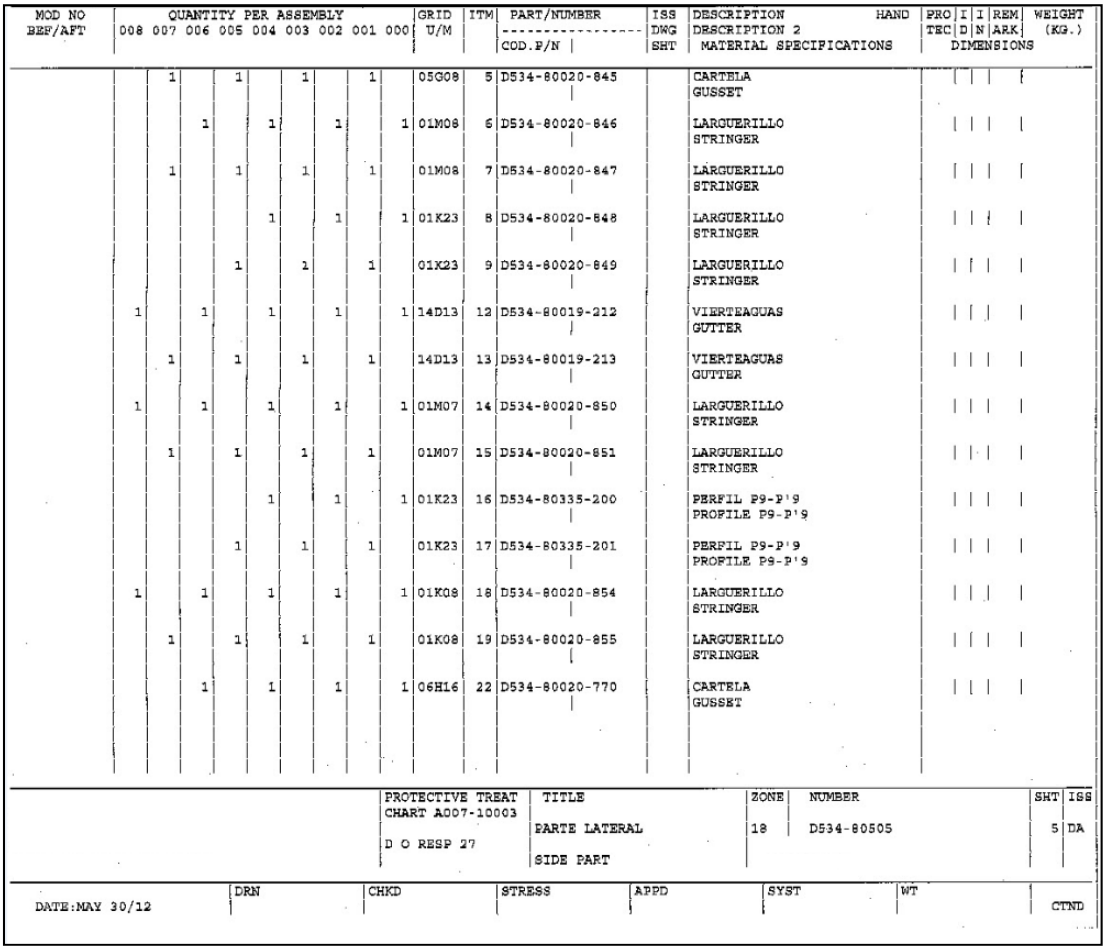
Заметим, что следует различать миграцию данных от интеграции данных. Миграция данных является мероприятием, в результате которого данные будут перемещены или скопированы из одной среды в другую, а затем удалены или списаны в источнике. Во время миграции (которая может продолжаться в течение нескольких месяцев или даже лет) данные могут передаваться в нескольких направлениях и может быть несколько процессов миграции, происходящих последовательно или одновременно. Необходимые действия по извлечению, преобразованию и загрузки данных будут совершаться здесь несколько отличными методами от тех, под совокупностью которых традиционно понимается ETL-процесс.

Экстракция данных. В подавляющем большинстве случаев унаследованные системы не имеют универсального интерфейса доступа к данным, используя который можно было бы получить доступ ко всем данным системы и перенести их на новое хранилище. Поэтому выбор способа экстракции данных является здесь ключевым. В рамках данной работы рассматривалась унаследованная PDM-система (англ. product data management — система управления данными об изделии) крупного авиационного предприятия. Было установлено, что PDM-система позволяет создавать бумажно-ориентированные документы со слабоструктурированными текстовыми данными (отчёты). Внешний вид отчётов представлен на рисунке 1.

Заметим, что работа с подобного вида документацией оказывается излишне трудозатратной и утомительной для персонала организации. Организация, использующая подобные устаревшие системы, вынуждена затрачивать дополнительные время и средства на обучение персонала, что удорожает общую стоимость владения информационной системой.

Трансформация данных. После анализа образцов документов, подобных представленному на рисунке 1, было выявлено, что структура документов не постоянна: тип выводимых данных, общее количество столбцов данных, ширина столбцов таблицы данных, информация в колонтитулах и другое может быть переменным. Отсюда следует, что при автоматизирован-

ной обработке данных из представленных документов требуется анализ контекста, окружающего эти данные. Отсюда встаёт задача автоматического анализа текстов.



MOD NO BEF/AFT	QUANTITY PER ASSEMBLY 008 007 006 005 004 003 002 001 000	GRID U/M	ITM	PART/NUMBER COD.P/N	ISS DWG SHT	DESCRIPTION HAND DESCRIPTION 2 MATERIAL SPECIFICATIONS	PRO TEC	I D	I N	REM ARK	WEIGHT (KG.) DIMENSIONS
	1 1 1 1	05G08	5	D534-80020-845		CARTELA GUSSET					
	1 1 1 1	01M08	6	D534-80020-846		LARGUERILLO STRINGER					
	1 1 1 1	01M08	7	D534-80020-847		LARGUERILLO STRINGER					
	1 1 1 1	01K23	8	D534-80020-848		LARGUERILLO STRINGER					
	1 1 1 1	01X23	9	D534-80020-849		LARGUERILLO STRINGER					
	1 1 1 1	14D13	12	D534-80019-212		VIERTEAGUAS GUTTER					
	1 1 1 1	14D13	13	D534-80019-213		VIERTEAGUAS GUTTER					
	1 1 1 1	01M07	14	D534-80020-850		LARGUERILLO STRINGER					
	1 1 1 1	01M07	15	D534-80020-851		LARGUERILLO STRINGER					
	1 1 1 1	01K23	16	D534-80335-200		PERFIL P9-P'9 PROFILE P9-P'9					
	1 1 1 1	01K23	17	D534-80335-201		PERFIL P9-P'9 PROFILE P9-P'9					
	1 1 1 1	01K08	18	D534-80020-854		LARGUERILLO STRINGER					
	1 1 1 1	01K08	19	D534-80020-855		LARGUERILLO STRINGER					
	1 1 1 1	06H16	22	D534-80020-770		CARTELA GUSSET					

PROTECTIVE TREAT CHART AD07-10003	TITLE	ZONE	NUMBER	SHT	ISS
D O RESP 27	PARTE LATERAL SIDE PART	18	D534-80505	5	DA

DATE:MAY 30/12	DRN	CHKD	STRESS	APPD	SYST	WT	CTND

Рис. 1. Образец экспортированного из унаследованной системы текстового документа

Анализ текстов в общем случае представляет собой сложную научно-техническую задачу, для решения которой существует множество различных методик (подробнее см., например, в работах [Christen, 2012; Borkar et al., 2001; Brants, 2002]). В данной работе анализ текстовых документов производился с помощью технологии регулярных выражений, основанной на теории формальных грамматик.

Основные положения теории формальных грамматик

Для того чтобы сделать более ясным дальнейшее изложение, вкратце напомним читателю основные положения теории формальных грамматик.

Будем называть алфавитом конечное непустое множество. Его элементы называются символами (буквами). Словом (цепочкой, строкой — от англ. string) в алфавите называется конечная последовательность элементов. Так, если задан алфавит $\Sigma = \{a, b, c\}$, тогда *baaa* является словом в алфавите Σ.

Слово, не содержащее ни одного символа (то есть последовательность длины 0), называется пустым словом и обозначается ε.

Длина слова w, обозначаемая $|w|$, есть число символов в w, причём каждый символ считается столько раз, сколько раз он встречается в w. Например, $|baaa| = 4$ и $|\varepsilon| = 0$.

Если x и y — слова в алфавите Σ, то слово xy (результат приписывания слова y в конец слова x) называется конкатенацией (катенацией, сцеплением) слов x и y. Иногда конкатенацию слов x и y обозначают $x \cdot y$.

Если x — слово и $n \in \mathbb{N}$, где $\mathbb{N}$ — множество натуральных чисел, то через x^n обозначается слово $\underbrace{x \cdot x \cdot x \cdot x ... \cdot x}_{n}$. По определению, $x^0 \rightleftharpoons \varepsilon$ (знак $\rightleftharpoons$ читается «равно по определению»).

Множество всех слов в алфавите Σ обозначается Σ^*, при этом множество всех непустых слов в алфавите Σ обозначается Σ^+. Так, если $\Sigma = \{a\}$, то $\Sigma^+ = \{a, aa, aaa, aaaa, ...\}$.

Говорят, что слово x — префикс (начало) слова y (обозначение $x \subset y$), если $y = xu$ для некоторого слова u. Очевидно, что $\varepsilon \subset baa$, $b \subset baa$, $ba \subset baa$ и $baa \subset baa$.

Говорят, что слово x — суффикс (конец) слова y (обозначение $x \supset y$), если $y = ux$ для некоторого слова u.

Говорят, что слово x — подслово (substring) слова y, если $y = uxv$ для некоторых слов u и v.

Через $|w|_a$ обозначается количество вхождений символа a в слово w. Так, например, если $\Sigma = \{a, b, c\}$, то $|baaa|_a = 3$, $|baaa|_b = 1|$ и $|baaa|_c = 0$.

Если $L \subseteq \Sigma^*$, то L называется языком (или формальным языком) над алфавитом Σ. Поскольку каждый язык является множеством, можно рассматривать операции объединения, пересечения и разности языков, заданных над одним и тем же алфавитом (обозначения $L_1 \bigcup L_2$, $L_1 \bigcap L_2$, $L_1 - L_2$). Так, множество $\{a, babb\}$ является языком над алфавитом $\{a, b\}$. Точно так же $\{a^k, ba^l | k \le l\}$ тоже будет являться языком над алфавитом $\{a, b\}$.

Пусть $L \subseteq \Sigma^*$. Тогда язык $\Sigma^* - L$ называется дополнением (complement) языка L относительно алфавита Σ. Когда из контекста ясно, о каком алфавите идёт речь, говорят просто, что язык $\Sigma^* - L$ является дополнением языка L.

Пусть $L_1, L_2 \subseteq \Sigma^*$. Тогда $L_1 \cdot L_2 \rightleftharpoons \{xy | x \in L_1, y \in L_2\}$. Язык $L_1 \cdot L_2$ называется конкатенацией языков L_1 и L_2, т. е. если $L_1 = \{a, abb\}$ и $L_2 = \{\text{bbc}, \text{c}\}$, то $L_1 \cdot L_2 = \{ac, abbc, abbbbc\}$.

Понятие автоматного языка. Понятие автоматного языка вытекает из теории конечных автоматов. Будем обозначать конечный автомат (англ. finite-state machine) пятёркой элементов:

$$M = \langle Q, \Sigma, \Delta, \text{I}, \text{F} \rangle,$$

где Σ — конечный алфавит, Q и Δ — конечные множества, причём $\Delta \subseteq Q \times \Sigma^* \times Q$, $I \subseteq Q$, $F \subseteq Q$.

Элементы Q — состояния (state), элементы I — начальные (initial) состояниями, элементы F — заключительные или допускающие (final, accepting) состояния.

Если $\langle p, x, q \rangle \in \Delta$, то $\langle p, x, q \rangle$ называется переходом (transition) из p в q, а слово x — меткой (label) этого перехода.

Путь (path) конечного автомата — это кортеж $\langle q_0, e_1, q_1, e_2, ..., q_n \rangle$, где $n \ge 0$ и для каждого i. При этом q_0 — начало пути, q_n — конец пути, n — длина пути, $w_1 ... w_n$ — метка пути.

Заметим, что для любого состояния $q \in Q$ существует путь $\langle q \rangle$. Его метка — ε, начало и конец совпадают.

Путь называется успешным, если его начало принадлежит I, а конец принадлежит F.

Слово w допускается (англ. — is accepted) конечным автоматом M, если оно является меткой некоторого успешного пути.

Язык, распознаваемый конечным автоматом M, — это язык $L(M)$, состоящий из меток всех успешных путей (то есть из всех допускаемых данным автоматом слов). Будем также говорить, что автомат M распознаёт (англ. — accepts) язык $L(M)$.

Понятие регулярного языка. Язык L называется автоматным (англ. — finite-state language), если существует конечный автомат, распознающий этот язык.

Будем называть обобщенным конечным автоматом аналог конечного автомата, где переходы помечены не словами, а регулярными выражениями. Метка пути такого автомата — произведение регулярных выражений на переходах данного пути. Слово w допускается обобщённым конечным автоматом, если оно принадлежит языку, задаваемому меткой некоторого успешного пути.

Каждый конечный автомат можно преобразовать в обобщённый конечный автомат, допускающий те же слова. Для этого достаточно заменить всюду в метках переходов пустое слово на 1, а каждое непустое слово на произведение его букв.

Если к обобщённому конечному автомату добавить переход с меткой 0, то множество допускаемых этим автоматом слов не изменится.

Язык L является регулярным тогда и только тогда, когда он является автоматным. Последнее заключение также известно как теорема Клини (подробнее см. [Пентус, 2004]).

Понятие регулярного выражения. Регулярное выражение над алфавитом $\sum$ определяется рекурсивно следующим образом: 0 является регулярным выражением; 1 является регулярным выражением; если $a \in \sum$, то a является регулярным выражением; если e и f являются регулярными выражениями, то $(e+f)$, $(e \cdot f)$ и e^* тоже являются регулярными выражениями. Условимся считать, что умножение связывает теснее, чем сложение. Вместо $e \cdot f$ часто пишут просто ef.

Например, пусть $\sum = \{a,b\}$. Тогда $((a \cdot b)^* \cdot (1+a))$ является регулярным выражением над алфавитом $\sum$.

Каждое регулярное выражение e над алфавитом $\sum$ задаёт некоторый язык над алфавитом $\sum$ (обозначение $L(e)$), определяемое рекурсивно следующим образом:

$$\begin{cases} L(a) \rightleftharpoons \{a\}, \text{если } a \in \Sigma; \\ L(0) \rightleftharpoons \varnothing; \\ L(1) \rightleftharpoons \varepsilon; \\ L(e+f) \rightleftharpoons L(e) \cup L(f); \\ L(e \cdot f) \rightleftharpoons L(e) \cdot L(f); \\ L(e^*) \rightleftharpoons L(e)^*. \end{cases}$$

Заметим, что в правой части последнего выражения символом «$*$» обозначена итерация языка.

Приведем пример на сказанное выше. Пусть $\sum = \{a,b\}$. Тогда согласно данному выше определению будет справедливо выражение

$$L((ab)^* \cdot (1+a)) = \{(ab)^n | n \geq 0\} \cup \{(ab)^n a | n \geq 0\}.$$

Введем понятие регулярного языка. Язык L называется регулярным, если он задаётся некоторым регулярным выражением. Поясним сказанное. Пусть e — регулярное выражение. Тогда

$$e^+ \rightleftharpoons e^* e.$$

Регулярные выражения — широко используемый способ описания шаблонов для поиска фрагментов текста и проверки их соответствия шаблону. По сути, регулярные выражения представляют собой систему обработки текста, основанную на специальной системе записи образцов (шаблонов) для поиска. Являясь мощным инструментом обработки произвольных текстов, используются многими текстовыми редакторами. Поддержка регулярных выражений реализована почти во всех современных языках программирования.

Регулярные выражения применяются для решения таких задач, как: а) проверка корректности тестового ввода пользователем; б) анализ текстовых данных с целью обнаружения структурных элементов (например, извлечение символьных данных из тэговой разметки); в) замена фрагментов текста, соответствующих заданному шаблону.

Методика миграции данных

На рисунке ниже представлена архитектура разработанного программного комплекса для практической реализации предлагаемого автором подхода.

Как видно из рисунка, описываемый комплекс представляет собой процесс наполнения базы данных структурированной информацией об изделии, с одной стороны, и обеспечение пользовательского доступа к этой информации через стандартизованный интерфейс доступа — с другой стороны.

Ниже в статье будет рассмотрен основные этапы показанной архитектуры.

Практическая реализация миграции данных

Для практической реализации предложенного метода был разработан программный комплекс на базе технологической платформы .NET Framework. Среди множества причин, по которым предпочтение было отдано .NET Framework, принципиальным было широкое использование на предприятии программных продуктов линейки Microsoft.

Поддержка регулярных выражений в .NET Framework. В платформе .NET Framework шаблоны регулярных выражений определяются с использованием синтаксиса, совместимого с регулярными выражениями Perl 5 и обладающего некоторыми дополнительными возможностями (например, сопоставление справа налево). Обработчик регулярных выражений платформы .NET Framework (предоставляется классом Regex) выполняет поиск с возвратом для регулярных выражений и является реализацией традиционной NFA-машины (недетерминированного конечного автомата) аналогично тем, которые используются в таких языках высокого уровня (ЯВУ), как Perl и Python, а также приложений Emacs и Tcl. Это отличает его от более быстрых, но и более ограниченных DFA-машин (детерминированный конечный автомат), предназначенных только для регулярных выражений и используемых в приложениях awk, egrep или lex. Это также отличает его от типовых, но более медленных POSIX NFA-машин.

Поддержка регулярных выражений в .NET выполняется классами пространства имен System.Text.RegularExpressions.Regex.

Алгоритм работы программного комплекса. Принципиальный алгоритм работы программного комплекса миграции данных с текстовым анализатором, представлен на рисунке 3. Наиболее важным этапом здесь является работа синтаксического анализатора. Суть его работы состоит в переборе заданных регулярных выражений для выделения из слабоструктурированного текста ценных данных. Определение ценных данных производится по метаданным, представленных в формате шаблонов регулярных выражений. Набор шаблонов регулярных выражений проектируются на основе экспертного анализа имеющихся образцов документов, после чего программист соответствующей квалификации должен спроектировать требуемые шаблоны.

Рассмотрим регулярное выражение для поиска даты выпуска документа. На рисунке 4 показана часть документа, с упоминанием даты выпуска в формате: «DATE<пробел>:<пробел>День(XX)<пробел>Месяц(XX)<пробел>Год(XXXX)».

Составим шаблон регулярного выражения на основе принятого в документе формата записи даты:

$$DATE\ \backslash s : \backslash s\ \backslash d\{2\}\ \backslash s\ \backslash d\{2\}\ \backslash s\ \backslash d\{4\},$$

здесь: $\backslash s$ — метасимвол, описывающий пробел; $\backslash d$ — цифровой символ; $\{n\}$ — оператор квантификации, определяющий, сколько раз предшествующее выражение может встречаться.

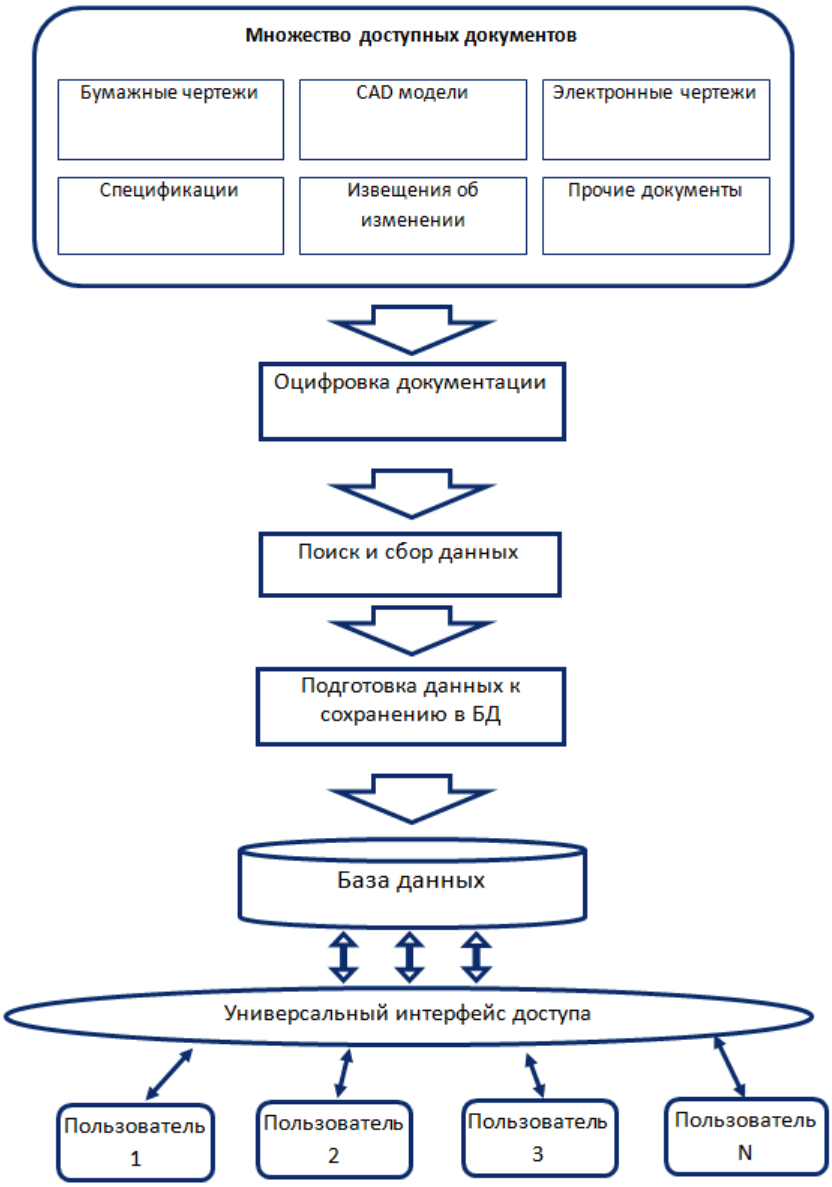


Рис. 2. Архитектура программного комплекса

Отрывок кода программы на языке C# представлен на рисунке ниже (см. рис. 5).

При вызове метод принимает в качестве параметра строковую переменную *line*.

В строке 119 создается экземпляр класса *Regex* с заданием в конструкторе 2 параметров: шаблона поиска, описанного выше и опции настройки обработчика регулярных выражений, в данном случае о разборе единственной строки.

В строке 120 запускается метод *Match* класса *Regex* для извлечения из текста одного первого вхождения набора символов, соответствующего шаблону регулярного выражения.

Метод *Match* возвращает объект *Match*, предоставляющий сведения о совпадении в тексте.

В строке 123 проверяется был ли поиск успешным и в случае успеха (условие *m.Success* возвращает логическое значение ИСТИННА) извлеченная строка приводится к формату типа *DateTime* платформы .NET. Приведение типов реализовано через вычленение подстроки (или суффикса, начиная с 7-го символа строки) извлеченной на предыдущем этапе строки.

В случае успешности приведения типов локальная переменная *foundDate* принимает значение даты выпуска рассматриваемого документа, после чего её значение может быть успешно транслировано и сохранено в базе данных.

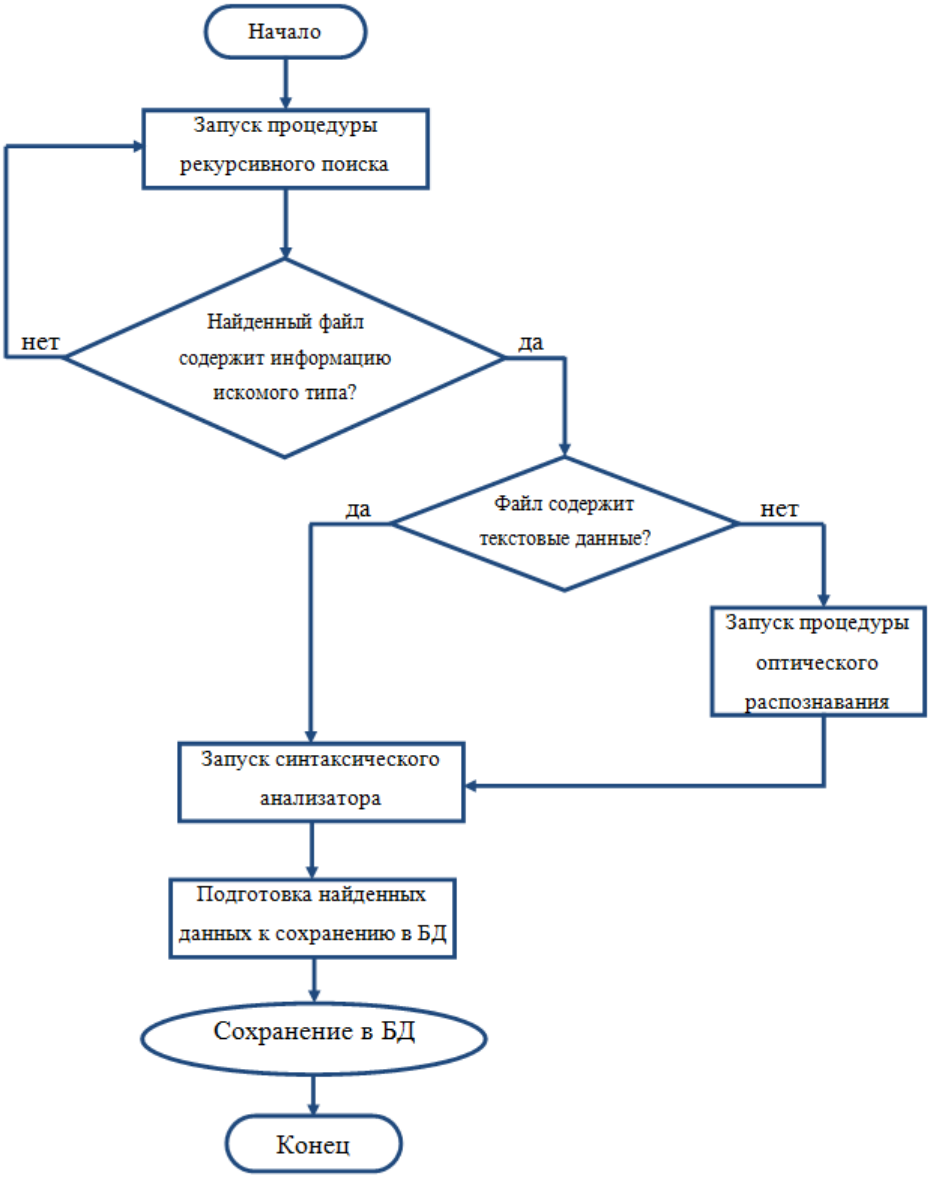


Рис. 3. Алгоритм работы программного комплекса миграции данных

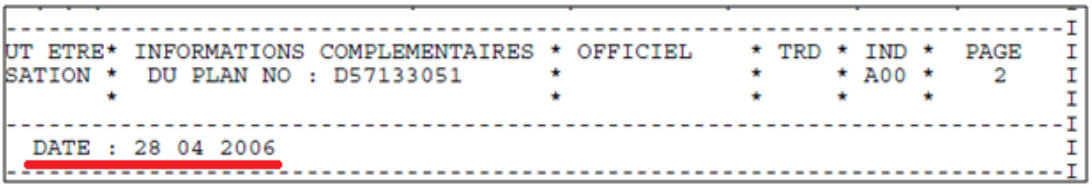


Рис. 4. Упоминание даты выпуска в экспортированном документе

После анализа текста с использованием необходимого набора шаблонов регулярных выражений выполняется проверка на целостность и непротиворечивость отобранных данных. Проверяется, все ли предусмотренные переменные получили значения требуемого типа, а также производится запрос к СУБД на предмет подтверждения уникальности обрабатываемого документа во избежание дублирования данных. В случае удовлетворительного результата выполняется этап трансформации данных.

```
117         private void parseReleaseDate(string line)
118         {
119             Regex re = new Regex(@"DATE\s:\s\d{2}\s\d{2}\s\d{4}", RegexOptions.Singleline);
120             Match m = re.Match(line);
121             DateTime foundDate = DateTime.Today;
122 
123             if (m.Success)
124             {
125                 if (DateTime.TryParse(m.Value.Substring(7),out foundDate))
126                 {
127                     releaseDate = foundDate;
128                 }
129             }
130         }
```

Рис. 5. Реализация метода для поиска даты выпуска документа

Трансформация данных. Назначение этапа трансформации данных в предлагаемом комплексе состоит в приведении типов отобранных из текстового документа данных к типам данных, поддерживаемых выбранной системы управления базами данных (СУБД).

Сохранение в БД. Для сохранения в целевой базе данных формируется SQL запрос типа INSERT, где аргументом выступает подготовленный набор данных, поддерживаемый целевой БД. На этом этапе вмешательство программиста или пользователя, как правило, не требуется.

Таким образом, ETL-процесс (Extract, Transform and Load) оказывается успешно завершенным.

Реализация доступа к данным

В качестве основы интерфейсного компонента, реализующего взаимодействие конечного пользователя и разработанной системы, использовалась библиотека Microsoft Active X Data Objects 6.0 Library (сокр. ADO).

ADO предоставляет прикладным системам набор функций для прямой работы с используемой в настоящей разработке базой данных SQL Server. Потенциальными пользователями библиотеки являются системы инженерных расчетов, офисные приложения пакета MS Office, системы электронного документооборота, CAD/CAM/CAE, PDM-, PLM-, ERP-, CAPP-, MES-системы и т. д.

Именно у этих систем возникает необходимость напрямую взаимодействовать с данными об изделии для получения первичной информации конструкторского проекта и при необходимости записи новой информации в проект.

Основное назначение модуля ADO — обеспечить программистов-разработчиков прикладных систем удобный доступ к данным, хранящихся в какой-либо БД. Получив доступ, прикладная система может отсылать запросы на нужную информацию. Данные, хранящиеся в восстановленной информационной модели, таким образом, становятся доступными многим популярным приложениям. А приложения, в свою очередь, пополняются мощным инструментарием работы с данными.

В качестве примера рассмотрим использование приложения MS Excel 2007 из пакета офисных приложений MS Office. Приложение MS Excel было выбрано по причине его относительной простоты эксплуатации, а также широкого распространения во многих коммерческих и государственных учреждениях России.

Возможности приложения MS Excel позволяют достаточно эффективно проводить научные, инженерные, экономические и статистические расчеты, пользуясь богатой встроенной библиотекой функций. В то же время приложение позволяет гибко манипулировать данными, используя в том числе доступ к удаленным базам данных.

Производителем пакета MS Office также распространяется набор инструментов для разработчиков «Средство разработки Microsoft Office в Visual Studio 2010», позволяющее создавать приложения для платформы .NET Framework, расширяющие пакет Microsoft Office. В рассмат-

риваемой работе использовалась библиотека *Microsoft.Office.Interop.Excel,* входящая в названный набор инструментов. Библиотека содержит набор интерфейсов для обеспечения взаимодействия между объектной моделью (COM) приложения Excel и сторонними приложениями.

На рисунке 6 показан внешний вид разработанного с использованием объектной модели Excel, элемента главного меню приложения. Как видно на рисунке 6, разработанный элемент управления полностью интегрирован в основное окно приложения, а расположенный на нем набор инструментов предоставляет пользователю обширные возможности для запроса и получения данных из БД.

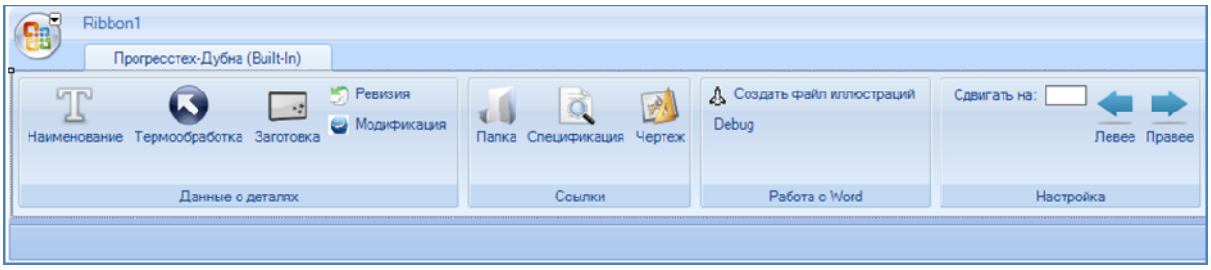


Рис. 6. Внешний вид пользовательского элемента главного меню приложения (ленты)

На рисунке 7 показан результат использования в работе описываемой надстройки Excel. Входными данными здесь являются набор номеров чертежей нескольких конструктивных элементов (выделено рамкой). После нажатия соответствующих кнопок на ленте меню в ячейках появляются запрошенные данные, такие как наименование детали, материал, заготовка, термообработка, ссылка на чертеж и другие.

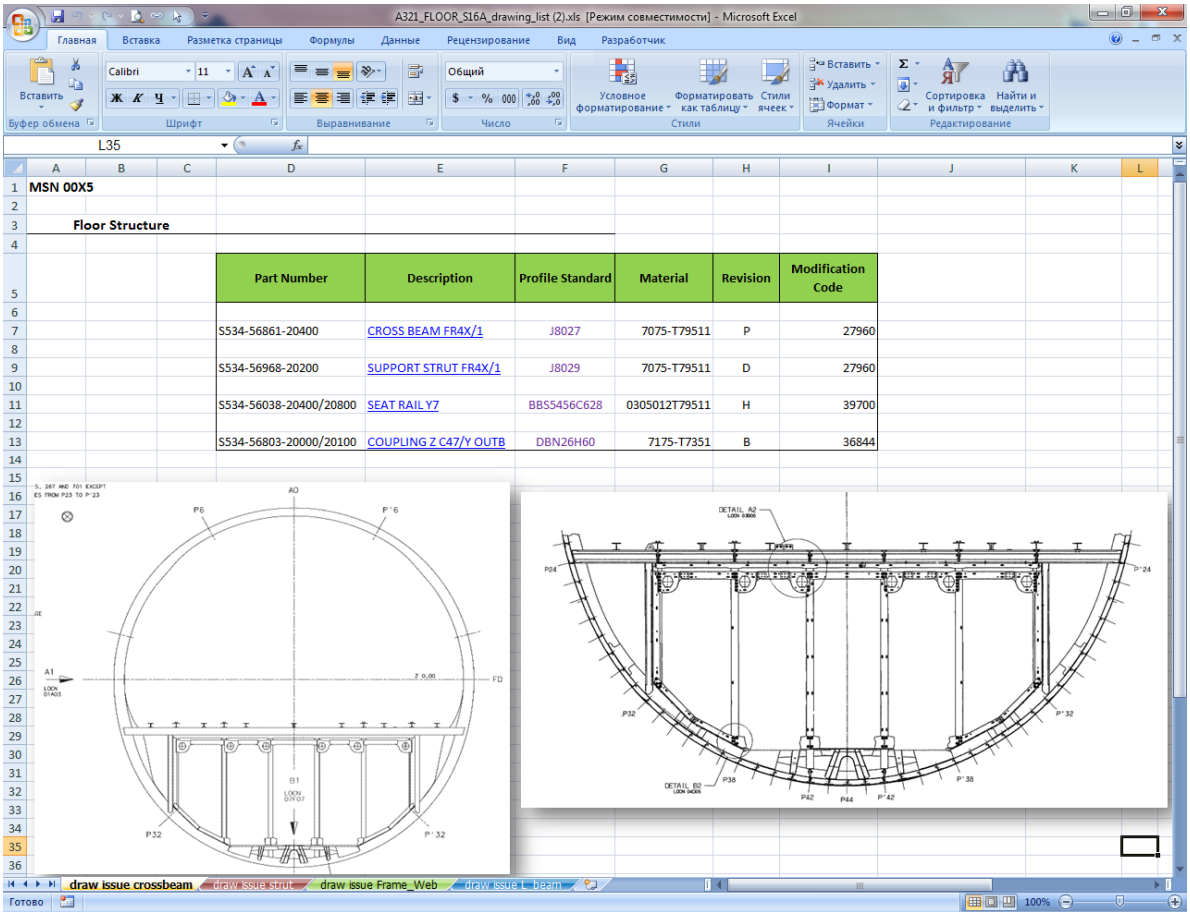


Рис. 7. Результат работы с надстройкой над Excel

Заключение

Автором предложена методика работы с унаследованными информационными системами, направленная на обеспечение доступности ценных данных за счет миграции данных с устаревшего ПО на современное программное и аппаратное обеспечение. Реализованное приложение позволяет автоматизировать работу широкого круга специалистов (инженеров, технологов, и т. д.), занятых в поддержке и модернизации образцов машиностроения. Предоставляет им возможности быстрого доступа к требуемым инженерным данным, исключая потери времени на ручной поиск документации. Приложение полностью интегрировано в MS Excel, что позволяет избежать затрат на обучение персонала.

Приложение было апробировано, внедрено и успешно используется в инженерной компании ООО «Прогресстех-Дубна» в рамках проектных работ по заказу крупнейших мировых авиационных производителей (Боинг, Эрбас и др.). Эффективность приложения подтверждается сокращением доли ручного труда инженера-конструктора и инженера-прочниста; повышением качества выпускаемой документации; экономии рабочего времени высококвалифицированных специалистов до 20–30 % ежемесячно.

Благодарности

Автор выражает благодарность своему научному руководителю профессору, д. т. н. В. В. Коренькову за ценные советы и замечания.

Список литературы

ГОСТ Р ИСО/МЭК 12207-2012. Информационная технология. Процессы жизненного цикла программных средств. — Введ. 01.03.2012 — М.: Госстандарт России, 2012.

ГОСТ Р ИСО/МЭК 15288-2005. Системная инженерия. Процессы жизненного цикла систем. — Введ. 29.12.2005. — М.: Федеральное агентство по техническому регулированию и метрологии, 2006.

Пентус А. Е., Пентус М. Р. Теория формальных языков / Учебное пособие. — М.: Изд-во ЦПИ при механико-математическом ф-те МГУ, 2004. — 80 с.

Тьюринг А. М. Вычислительные машины и разум. — Самара: Бахрах-М, 2003.

Borkar V., Deshmukh K., and Sarawagi S. Automatic segmentation of text into structured records, 2001.

Brants T. Topic-based document segmentation with probabilistic latent semantic analysis. In Proceedings of CIKM (McLean, pages 211–218). ACM Press, 2002.

Christen P. A survey of indexing techniques for scalable record linkage and deduplication. IEEE Transactions on Knowledge and Data Engineering, 24: 1537–1555, 2012.

Heine J. Retiring old applications: reduce IT cost through demand management principles. Application Management, Gartner, Inc, 10.11.2006, с. 4.

McCormick J. Mainframe-web middleware [электронный ресурс] // GCN. — 2013. — URL: http://gcn.com/articles/2000/06/02/mainframeweb-middleware.aspx (дата обращения: 06.02.2013).

Middleware (distributed applications) [электронный ресурс] // Wikipedia. — 2013. — URL: http://en.wikipedia.org/wiki/Middleware_%28distributed_applications%29 (дата обращения: 06.02.2013).

Murphy P. Got Legacy? Four fates await your applications // Trends, Forrester Research, 10.01.2006, с. 2.