УДК 004.89

А.Л. Ерохин, Р.А. Полунин

ОБ ИНТЕЛЛЕКТУАЛИЗАЦИИ ПРОЦЕССА ПОДДЕРЖКИ ЖИЗНЕННОГО ЦИКЛА ПРОГРАММНЫХ СИСТЕМ

1. Введение

В процессе поддержки жизненного цикла (ЖЦ) программных систем (ПС) важную роль играет информационное обеспечение. Термин «информационное обеспечение» заимствован в сферу информационных технологий из теории АСУ, сама по себе поддержка ЖЦ больших коммерческих ПС на сегодня является достаточно сложным процессом. В широком смысле под информационным обеспечением системы понимают создание информационных условий функционирования системы, обеспечение необходимой информацией, включение в систему средств поиска, получения, хранения, накопления, передачи, обработки информации, организацию банков данных. Под информационным обеспечением (ИО) в задачах, входящих в процесс поддержки ЖЦ ПС, например моделирования, проектирования и разработки, обычно понимают комплекс мер по организации создания, хранения и поиска тех или иных информационных ресурсов, имеющих отношение к ЖЦ ПС. Примерами таких ресурсов могут служить различные формальные и неформальные информационные модели (например диаграммы в нотациях UML, IDEF), специализированные базы данных (сведения о логической и других структурах конкретной ПС, о взаимозависимостях ее подсистем, временные планы работ по задачам разработки), и, что самое важное, различного рода информация, которая может быть автоматически или полуавтоматически получена из исходных кодов ПС.

В настоящее время задачи ИО поддержки ЖЦ ПС решаются исключительно в контексте автоматизации тех или иных стандартизованных процессов разработки, таких как RUP (Rational Unified Process). Постановка решаемых задач ИО логически вытекает из структуры этих процессов и преследуемых целей: повышение качества кода, рефакторинг, тестирование. Однако эти задачи пока что решаются на прикладном уровне, без привлечения современных подходов к управлению знаниями, таких как построение баз знаний, автоматизация семантического поиска и т.д.

С другой стороны, существует широкий спектр алгоритмов и общих подходов к семантическому поиску в текстовых и других источниках плохо структурированной информации, применяемых в современных поисковых системах. Все многообразие моделей традиционного информационного поиска принято делить на три вида: теоретико-множественные (булевская, нечетких множеств, расширенная булевская), алгебраические (векторная, обобщенная векторная, латентно-семантическая, нейросетевая) и вероятностные [1, 2]. Предметом поиска в поисковых системах выступает информационная потребность пользователя, выраженная в запросе. В рамках *информационного поиска* изучаются все составляющие процесса поиска, а именно: предварительная обработка информации (индексирование), обработка и исполнение запроса, ранжирование, пользовательский интерфейс и обратная связь.

2. Постановка задачи исследования

До недавнего времени задачи информационного обеспечения поддержки жизненного цикла (ЖЦ) ПС рассматривались исключительно в рамках стандартизированных процессов разработки (таких как SADT, SSADM, RUP, SCRUM, eXtreme Programming). Это автоматически сводило рассматриваемую проблематику к тем задачам, которые возникали в этих процессах, например автоматизация построения моделей и генерирования кода (направление CASE-средств), управление версиями исходных кодов, автоматизация сопровождения ПС, рефакторинга. Однако развитие систем управления знаниями (knowledge management) и информационного поиска (information retrieval) открывает новые возможности интеллектуализации ИО.

Предметом настоящего исследования является интеллектуализация процесса информационного обеспечения в контексте поддержки жизненного цикла (ЖЦ) программных систем (ПС).

Целью исследования является разработка такого подхода к ИО ЖЦ ПС, который позволил бы управлять степенью формализации информационной модели ПС в зависимости от поставленных аналитиками-пользователями целей.

Наиболее активно развивающаяся сегодня концепция формализации процессов разработки ПС (model-driven architecture; архитектура, основанная на моделях) основывается на априорном моделировании ПС с помощью различных CASE-средств. Успешное применение такого подхода требует выполнения определенных требований, таких как высокая квалификация проектировщиков, высокий уровень формализации требований к ПС и организации процесса разработки. К сожалению, на практике эти требования очень часто хотя и выполнимы, но далеко не в той степени, чтобы сделать

возможным полный переход к процессам разработки ПС, основанным на моделях. Кроме того, существует большое поле задач, связанных с анализом уже созданных коммерческих ПС, для которых автоматизация построения моделей (например с помощью *обратного проектирования*) крайне затруднена. Этими факторами и обусловлена **актуальность исследования**.

В связи с этим для существующей ПС целесообразно несколько снизить требования к степени формализации (для упрощения разработки моделей) и всеохватности строящихся по ней моделей (для ускорения их автоматического построения для данной ПС). При решении прикладных задач анализа ПС компенсировать увеличение количества моделей и снижение их степени формализации можно за счет применения принципов и технологий информационного поиска (information retrieval) и поиска данных (data retrieval). Теоретические основы информационного поиска были заложены в 60–70-х годах (Swets, Rijsbergen, Robertson, Good, Cooper) и с тех пор активно разрабатываются ведущими исследовательскими группами и активно применяются как для поиска в глобальных и локальных сетях и базах данных, так и для организации, например, рефакторинга программного кода.

Идеи алгоритма поиска, наиболее близкие к нашей задаче, изложены в [3, 4] и развиты, например, в [5]. Суть [3] заключается в том, чтобы применить для поиска распространяющуюся по ребрам графа волну. Объем набора результатов будет зависеть от начального уровня «энергии» и весовых коэффициентов ребер, релевантность каждого элемента результата – количеством дошедшей до него «энергии». Основным недостатком такого подхода является необходимость непосредственного хранения в базе знаний всех ребер, которые используются при распространении «волны».

Для решения поставленной цели рассмотрим новый подход к интеллектуализации задач информационного обеспечения (ИО), который ориентирован на информационный поиск и поиск данных.

3. Подход к интеллектуализации ИО

3.1. Модель процесса ИО

В данной статье информационное сопровождение процесса разработки ПО рассматривается как отдельный процесс. Следовательно, необходимо сначала описать используемую модель процесса информационного обеспечения. Информационная модель процесса ИО, используемая в данной работе, представлена на рис. 1.

Под блоком №1 подразумеваются различные среды разработки приложений, управления версиями, репозитарии документов и т. п., в которых и с помощью которых ведется проектирование и разработка (в том числе формализованные) программного продукта.

Рис. 1. Структура рассматриваемой части процесса ИО

Блок №2 включает в себя две основных деятельности, описанные выше: наполнение базы знаний и выполнение запросов к ней.

Под агентами сбора информации подразумеваются программные модули, автоматически и полуавтоматически извлекающие различные сведения непосредственно из исходных кодов программ либо путем мониторинга их во время выполнения. Эти сведения могут включать в себя: физическую структуру проектов, перечни классов/глобальных переменных/процедур и прочих языкозависимых элементов, использование определенных элементов одной среды (как процедур в БД) элементами другой среды (клиентским кодом приложения) и др. Кроме того, в эти сведения могут входить и логические структуры проектов по подсистемам и задачам, номера версий продукта, перечни сотрудников, связи сотрудников с подсистемами и др., которые могут быть введены вручную или импортированы из имеющихся источников.

Агенты сбора информации выполняют функцию наполнения и обновления БЗ, находящейся под управлением некоторого сервера. Аналитик, которому требуется выполнить запрос к этой БЗ (например получить перечень логических подсистем продукта, которые нужно протестировать после изменения определенного модуля), также обращается к серверу, используя некий формальный язык.

Настройки, которые влияют на способы формирования ответа на запрос, определяются Администратором БЗ, который отвечает за построение

онтологической схемы БЗ, выбор алгоритмов обхода конкретных типов зависимостей (при выполнении запроса) и за поддержку работы агентов сбора информации.

Далее определим понятие базы знаний процесса поддержки ЖЦ ПС, используемой в процессе ИО, и рассмотрим её составляющие.

3.2. База знаний процесса информационного обеспечения поддержки жизненного цикла ПС (БЗП)

База знаний процесса информационного обеспечения поддержки жизненного цикла ПС – это множество текстовых и других ресурсов, содержащих структурированную и формализуемую информацию о различных аспектах проекта: деревья иерархий классов, взаимозависимости модулей, включение модулей в проекты, использование классов в модулях, логическая структура проекта по подсистемам, связи с документацией и т. п.

Эта информация часто представима в виде деревьев, ориентированных и неориентированных графов и т. д. Можно понимать элементы БЗП как множество объектов $X = \{xi\}$ и множества признаков L, с помощью которых формируется представление этих объектов в ЭВМ. Каждый из признаков задает наличие у данного объекта некой характеристики. То есть он выступает в описываемой этим признаком *роли*. Чтобы представить БЗП в формальном виде, т. е. извлечь ее формальную структуру из разнородных источников, необходимо разработать онтологическую схему и построить средства извлечения информации из имеющихся источников под данную онтологию.

В то же время есть необходимость в том, чтобы при разработке подобной базы знаний уйти от непосредственного использования термина «объект», ограничившись лишь различными *описаниями* объектов. Эта необходимость обусловлена двумя причинами: 1) источники формализованной информации об отношениях элементов системы часто независимы друг от друга и, как следствие, не всегда возможно составить единый перечень объектов для решения всех задач ИО очередной ПС; 2) в данной работе рассматривается в первую очередь задача применения различных видов поиска для построения описаний некоторого термина или группы терминов в заданном контексте, а также построение семантически связанных групп элементов.

3.3. Формализация определения БЗП

БЗП определим формально как тройку (LBL, REL, CUT), где LBL – система меток, REL – множество некоторых топологически формализуемых отношений и CUT – множество так называемых срезов БЗ, реализующих эти отношения для данного процесса разработки.

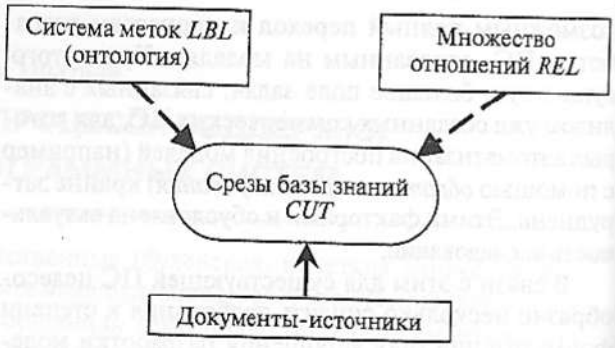

Рис. 2. Элементы базы знаний процесса ИО

Система меток, словарь отношений и срезы определяются спецификой конкретной ПС, доступными источниками информации. Множество отношений REL образует структуру БЗП. Это только те отношения, которые возможно полностью формализовать, причем автоматически или полуавтоматически. Последнее требование связано с тем, что эти отношения часто содержат очень большое количество элементов. Сами по себе эти отношения могут быть как специфическими для данного процесса ИО или конкретной ПС, так и общими для многих ПС.

Система меток в базе знаний процесса ИО выполняет онтологическую функцию. Множество срезов CUT представляет собой содержимое базы знаний данного процесса ИО, формализованной в терминах системы меток LBL через отношения REL. Наполнение срезов и поддержка их в актуальном состоянии происходит с помощью агентов сбора информации параллельно с модификацией самой базы знаний процесса программистами, документалистами и аналитиками (см. рис. 1).

3.4. Определение системы меток

Систему меток LBL определим следующим образом:

$LBL = (L, RG, \omega_{расш}, \omega_{суж})$,

L – множество меток,

$RG = (N, RR)$ – граф семантических связей,

где N – множество вершин,

RR – множество ребер,

$\omega_{расш}, \omega_{суж}$ – стоимости расширения и сужения контекста,

$RR = \{(l_1, l_2, \omega_1, \omega_2)\}$, l_1, l_2 – метки,

ω_1, ω_2 - стоимости перехода.

Граф связей системы меток имеет следующие свойства:

1) является семантической сетью, в которой имеются как однонаправленные, так и двунаправленные связи (это зависит от величин стоимостей перехода);

2) может быть многокомпонентным;
3) не имеет кратных ребер.

С каждой связью в графе системы меток сопоставим два числа $\omega_1, \omega_2 \in [0;1]$. Эти числа будем называть *стоимостями перехода* между метками данной связи. Если стоимость перехода в заданном направлении равна нулю, то будем говорить, что такой переход невозможен. Чем ближе стоимость перехода к 1, тем более эти метки связны в данном направлении перехода.

Связи в системе меток можно классифицировать по соотношению стоимостей прямого и обратного переходов:

— тождественная: $\omega_1 = \omega_2 = 1$;
— полутождественная: $\omega_1 \neq \omega_2$; ω_1 или $\omega_2 = 1$;
— синонимическая: $\omega_1 = \omega_2$; $0 < \omega_1 < 1; 0 < \omega_2 < 1$;
— общего вида: $\omega_1 \neq \omega_2$; $0 \le \omega_1 < 1; 0 \le \omega_2 < 1$.

3.5. Определение переходов и связности меток

Если в системе меток есть связь $rr(l_1, l_2, \omega_1, \omega_2)$, то прямым переходом по данной связи будем называть переход от метки l_1 к метке l_2, а *обратным*, соответственно, от l_2 к l_1.

Будем говорить, что связность перехода от произвольной метки l к этой же метке l (петля в графе) всегда равна 1 для любой системы меток.

В более общем случае переход от одной метки к другой — это поиск такой цепочки переходов между этими метками, связность которой будет больше нуля.

Определим *цепочку перехода* между l_1 и l_2 как *цепь* между l_1 и l_2 (т.е. *маршрут*, все ребра которого различны) в графе системы меток, имеющую ненулевую *связность*.

$$rc(l_1, l_2) = \{(rr_i, \omega_i^k)\},$$

ω_i^k — стоимость перехода по связи rr_i

$$k = \begin{cases} 1, \text{если это прямой переход} \\ 2, \text{если это обратный переход} \end{cases}$$

Связность σ цепочки перехода rc вычисляется по формуле

$$\sigma(rc) = \prod_i \omega_i, \text{ причем } \sigma(rc) > 0,$$

где ω_i — стоимость i-го перехода цепочки

Итак, возможность построения цепочки перехода между двумя метками определяется двумя факторами: принадлежностью этих меток одной компоненте графа связей и расстановкой весов связей в этом графе.

Поскольку цепочек перехода между двумя метками в данной системе меток может быть более одной, то определим *лучшую цепочку* перехода между двумя метками как одну из цепочек с наибольшей связностью во множестве всевозможных цепочек:

$$RC(l_1, l_2) = \{rc(l_1, l_2)\},$$

лучшая цепочка $rcm \in RC$, $\sigma(rcm) = \max$.

Интересного эффекта можно добиться, используя значения стоимостей, большие 1: можно *увеличивать* глубину, на которую поисковый алгоритм будет раскрывать связи в срезах БЗ, встречая определенные метки.

3.6. Определение унификации меток и групп меток

Две метки l_1, l_2 будем называть *унифицируемыми* со связностью σ, если $RC(l_1, l_2) \neq \varnothing$, где $\sigma = \sigma(rcm)$, rcm — лучшая цепочка в RC.

Группой меток назовем непустое упорядоченное подмножество L.

Группы меток *равны*, если они имеют одинаковый размер, и метки в соответствующих позициях совпадают.

Определим *интерпретацию группы* L как одну из перестановок L.

В общем случае две группы меток L_1 *и* L_2 будем называть унифицируемыми со связностью σ, если существует хотя бы одна интерпретация L_1^* и L_2^*, при которых $\sigma(L_1^*, L_2^*) > 0$. Но, поскольку таких интерпретаций может быть несколько, будем понимать унификацию L_1 *и* L_2 как поиск и выбор одной из интерпретаций, при которых связность максимальна.

Правило вычисления связности двух групп меток (или их перестановок) L_1 и L_2, имеющих одинаковую размерность, запишем как:

$$\sigma(L_1, L_2) = \prod_{i,j} \sigma(l_i, l_j), \text{ где } l_i \in L_1,\ l_j \in L_2,\ i = j,$$

ясно, что $\sigma(L_1, L_2) = 1$, если $L_1 = L_2$.

В общем случае $\sigma(L_1, L_2) \neq \sigma(L_2, L_1)$, т.е. операция вычисления связности некоммутативна.

Если L_1 *и* L_2 имеют различное количество меток, то говорим, что происходит расширение или сужение контекста. При этом из большей группы для вычисления $\sigma(L_1, L_2)$ необходимо выбрать такое подмножество меток, равное по размеру меньшей группе, чтобы одна из перестановок этого подмножества максимизировала $\sigma(L_1, L_2)$.

Будем говорить, что если размерность L_1 больше размерности L_2, то имеет место *сужение* контекста. А если количество меток в L_1 меньше, чем в L_2, то имеет место *расширение* контекста. С расширением и сужением контекста также связаны определенные стоимости, величины которых задаются константами $\omega_{\text{расш}}$, $\omega_{\text{суж}}$, общими для системы меток, используемой при унификации данных групп меток. Эти стоимости включаются в виде множителей в общее произведение в формуле вычисления связности $\sigma(L_1, L_2)$.

Таким образом, на величину связности двух групп меток влияют два фактора: стоимости переходов, сопоставленные со связями системы меток, и стоимости расширения/сужения контекста, указанные в определении системы меток.

Приведем выражение для подсчета связности групп меток с учетом различного количества их элементов:

$$\sigma(L_1, L_2) = \prod_{i,j} \sigma(l_i, l_j) * \omega^m,$$

где $l_i \in L_1$; $l_j \in L_2$; $i = j$; m – разница в количестве элементов;

$$\omega = \begin{cases} \omega_{\text{расш}}, \text{если количество элементов } L_2 \text{ больше } L_1 \\ \omega_{\text{суж}}, \text{если количество элементов } L_1 \text{ больше } L_2. \end{cases}$$

3.7. Определение срезов базы знаний

Структура БЗП определяется множеством отношений $R \in REL$, которые представлены в ней соответствующими *срезами*. Определим срез базы знаний процесса разработки как ориентированный граф, одно- или многокомпонентный, узлы которого связаны друг с другом отношением R. Срезы БЗ могут быть построены по топологически формализуемым отношениям либо автоматически, либо вручную. Примерами срезов БЗ могут служить: лес иерархий классов в исходном коде, дерево зависимостей объектов в базе данных, логическое разбиение проекта на подсистемы.

Определим срез формально (*cut*):

$$cut = (Name, R, N, L, M), cut \in CUT, R \in REL,$$

где $Name$ – имя среза;

R – отношение, по которому построен граф среза;

N – множество узлов (вершин графа среза);

L – множество связей (ребер графа среза), реализующих отношение R, между узлами из N; $L = \{l_i\}, l_i = (n_1, n_2)$.

Таким образом, срез представляет собой одно- или многокомпонентный ориентированный граф, связи которого отвечают отношению R. В базе знаний для каждого отношения может содержаться более одного среза. Объединение всех срезов (построенных по отношению R) эквивалентно самому отношению R.

Узлом среза будем называть вершину графа среза, с которой сопоставлена *сигнатура*, т. е. пара $(Name, M)$, где $Name$ – имя данного узла, M – группа меток, обязательно содержащая все элементы образующего множества меток данного среза. Узел среза не может быть однозначно идентифицирован своей сигнатурой. Имя узла несет смысловую нагрузку, непосредственно связанную с тем процессом разработки, в рамках которого строится БЗП.

3.8. Определение поиска по БЗП

По аналогии с [3] определим задачу *поиска* по базе знаний процесса разработки ПС как задачу построения структурированного множества узлов одного или более срезов БЗ, входными данными которой является сама БЗ и *опорное множество* сигнатур узлов $\{s_i\}$, где s_i – сигнатура узла.

Поиск заключается в пошаговом расширении протокола поиска (структурированного множества узлов и информации о ходе поиска) за счет выполнения двух видов *переходов*: а) экстенсивного, б) интенсивного.

Экстенсивным переходом $Ex(n, n_2)$ назовем выбор связи $l(n, n_2)$ в графе среза, которому принадлежит n, и добавление в рабочий набор узла n_2. Экстенсивный переход осуществляется за счет наличия в срезе базы знаний непосредственной связи между узлами. Эта связь отражает отношение R, которое присутствует в сигнатуре данного среза.

Интенсивным переходом $In(n, n_2)$ назовем поиск такого узла n_2, который унифицируется с узлом n, и добавление его в рабочий набор. Особенностью интенсивного перехода является то, что n_2 может принадлежать любому срезу, а не только тому, которому принадлежит n. Интенсивный переход осуществляется за счет наличия косвенной связи между группами меток через систему меток.

С каждым узлом, записываемым в протокол поиска, ассоциируется некоторая величина связности, которая должна быть больше или равной некоторому нижнему пределу в диапазоне (0; 1]. Это число позволит ограничить «чувствительность» поиска, то есть длину цепочек, выводимых от элементов опорного множества.

3.9. Алгоритм поиска

Входные данные: опорное множество сигнатур (групп меток) и нижний предел связности.

Выходные данные: многоуровневый протокол поиска.

Порядок работы:

1. Инициировать пустой протокол поиска.
2. Считать текущий уровень нулевым.
3. Построить базу поиска и добавить ее элементы в протокол на текущем уровне.
4. Увеличить номер текущего уровня.
5. В текущий уровень занести множество всех узлов, которые унифицируемы (со связностью, большей нижнего предела) хотя бы с одним узлом предыдущего уровня и еще не содержатся в протоколе поиска. Если узел может быть получен в результате перехода с более чем одного узла предыдущего уровня, величина связности каждого добавляемого узла выбирается наибольшей из всех возможных.
6. Если в текущем уровне есть хотя бы один узел, вернуться к шагу 4.

Иллюстрация возможного протокола поиска приведена на рис. 3.

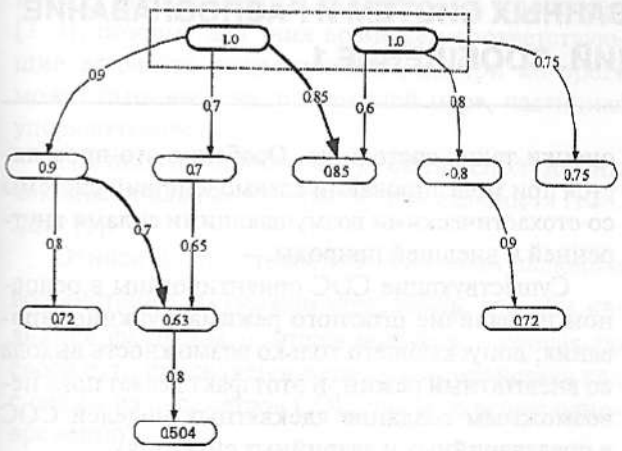

Рис. 3. Протокол поиска

На рис. 3 показаны узлы и переходы, выполненные при поиске. Числами указаны стоимости переходов и уровни связности цепочки в каждом узле-результате, пунктиром выделено опорное множество узлов. В показанном протоколе поиска имеются 4 уровня (горизонтальные группы узлов сверху вниз), от 0-го до 3-го. На нулевом уровне находится *база поиска* (строим по опорному множеству).

В начале работы алгоритма поиска протокол заполняется узлами, сигнатуры которых унифицируемы хотя бы с одной сигнатурой из опорного множества. Будем называть это первичное множество узлов *базой поиска*. Для каждого элемента базы поиска устанавливается величина связности, равная наибольшей из связностей сигнатуры этого узла с сигнатурами опорного множества. При расширении протокола поиска (путем интенсивных и экстенсивных переходов) новым добавляемым узлам будет сопоставлена величина связности, равная произведению связности исходного узла и стоимости соответствующего перехода.

При выполнении каждого перехода происходит понижение связности, вплоть до некоторого заданного минимума. Это позволит контролировать глубину поиска. С другой стороны, ширину поиска, то есть охват срезов БЗ, в принципе возможно контролировать путем установления нижней границы связности для каждого среза БЗ отдельно, например путем нормирования нижней границы по количеству узлов каждого среза.

3.10. Нижние пределы связности и унифицируемости

Унификация групп меток производится при поиске узлов для выполнения интенсивного перехода. В начале построения очередной ветви цепочки поиска от опорного множества алгоритм устанавливает текущий уровень связности в 1. Каждый интенсивный переход между узлами увеличивает длину цепочки поиска и уменьшает уровень связности путем домножения его на стоимость очередного перехода. Как только уровень связности падает ниже некоторой заданной константы, развитие данной цепочки прекращается. Эту константу назовем *нижним пределом связности цепочки*.

Кроме того, если связность групп меток узлов ниже некоторой константы, то интенсивный переход между ними считается невозможным. Эту константу назовем *нижним пределом унифицируемости групп меток*.

4. Заключение

В настоящее время появляется необходимость в максимально гибких средствах анализа информации, создаваемой в процессе поддержки ЖЦ программных систем, однако на пути создания таких гибких средств стоит проблема формализации источников и сложности строящихся моделей. Одним из возможных решений может стать предлагаемое снижение степени формализации информационной базы ЖЦ ПС и даже создание возможности управления степенью ее формализации за счет применения поисковых технологий.

Таким образом, в данной работе обоснован новый подход к интеллектуализации задач информационного обеспечения, предложены новые модели и алгоритм информационного поиска.

В настоящей работе кратко рассмотрен лишь один из возможных алгоритмов поиска по предлагаемой структуре данных. Направлением дальнейших исследований является разработка методов и алгоритмов автоматизации анализа исходных кодов с использованием базы знаний процесса ИО, основанной на системе меток и срезах. Необходимо также рассмотреть возможности «обучения» БЗП, оптимизации запросов.

Список литературы: 1. *Baezo-Yates R., Ribeiro-Neto B.* Modern Information Retrieval. ACM Press, Addison Wesley. 1999. 2. *C. J. van RIJSBERGEN.* INFORMATION RETRIEVAL, 2nd edition. 1979. 3. *Preece, Scott.* A spreading activation network model for information retrieval. PhD thesis, CS Dept., Univ. of Illinois, 1981. 4. *Cohen P., Kjeldsen R.* Information Retrieval by Constrained Spreading Activation on Semantic Networks // Information Processing and Management. 1987. 23(4) C. 255-268. 5. *Maciej Ceglowski, Aaron Coburn, John Cuadrado.* Semantic Search of Unstructured Data using Contextual Network Graphs // National Institute for Technology and Liberal Education, 2003.Vermont, 05753 USA. 6. *ГОСТ 24.205-80.* Требования к содержанию документов по информационному обеспечению.

Поступила в редколлегию 21.11.2004