Комплексное решение проблемы 2000 года в корпоративных информационных системах

Профессор А.Н.Терехов, А.А.Терехов, В.В.Уфнаровский
ЗАО "ЛАНИТ-ТЕРКОМ",
Санкт-Петербургский Государственный Университет
Опубликовано в журнале "Информатизация и связь", №1, 1999

Масштаб проблемы 2000 года

Сейчас, когда до смены тысячелетий осталось всего лишь чуть больше года, уже никто не сомневается в существовании проблемы 2000 года. Даже в России, наконец, осознают опасности и сложности, связанные с устаревшими приложениями, от которых зачастую зависят целые предприятия, банки и критические системы реального времени, такие как управление самолетами.

Приведем некоторые независимые оценки масштабов проблемы 2000 года. Известная консалтинговая фирма Gartner Group приводит следующие прогнозы:

- решение проблемы 2000 года поглотит до 30% бюджета на информационные технологии всех компаний;
- 23% компаний еще не предприняли никаких усилий по решению проблемы 2000 года;
- успешно решат проблему проблему 2000 года только 50% компаний, имеющих информацонные системы.

Дополнением к последнему прогнозу звучит утверждение британской консалтинговой фирмы META Group, что до 50% компаний столкнутся в своих критических для бизнеса системах с ошибками, вызванными проблемой 2000 года. Наконец, Комиссия Сената США оценила затраты на решение проблемы 2000 года в госструктурах в 50 миллиардов долларов.

Отметим также оценки масштаба проблемы 2000 года, приводимые в научных статьях. Например, в статье [4], глобальные затраты на решение проблемы 2000 года оцениваются в 1.34 триллиона долларов (из них 530 миллиардов только на исправление программ). Другим показателем является оценка в 30% от ежегодного бюджета всех предприятий, занимающихся информационными системами [3].

Поэтому сейчас многие организации начинают предпринимать срочные попытки решения проблемы 2000 года, вкладывая деньги в изготовление

"заплаток" на программных комплексах. На алтарь решения проблемы 2000 года без малейших колебаний возлагаются деньги, время и сотрудники.

Все это уже было в США и других западных странах, причем на несколько лет раньше. Какие выводы мы можем сделать на основе чужого опыта, повторения каких ошибок мы можем избежать? Как можно справиться с проблемой 2000 года и потратить на этом наименьшее количество средств? Такие вопросы стоят в данный момент перед многими руководителями предприятий и начальниками отделов информационных технологий. В данной статье мы попытаемся сформулировать некоторые ответы.

Особенности проблемы 2000 года в России

Нет худа без добра – относительно невысокий уровень информатизации плюс наличие большого количества высококвалифицированных программистов облегчают решение проблемы 2000 года в России. Это, в частности, отражено в выкладках Государственного комитета по информатизации и связи, где уровень информатизации России принимается за 25% от уровня США.

С другой стороны, к дополнительным сложностям проблемы 2000 года в России следует отнести следующие факторы:

- запоздалое осознание проблемы в России по сравнению с западными странами;
- большое количество нелегально используемого системного программного обеспечения;
- нежелание российских руководителей предпринимать конкретные действия;
- другой набор используемых языков и платформ по сравнению с Западом;
- отсутствие у организаций денег на решение проблемы 2000 года.

Наш опыт в области проблемы 2000 года

Уже более шести лет мы работаем в области проблемы 2000 года и связанных с ней проблем евровалюты и реинжиниринга. Сначала мы сосредоточились на разработке программного обеспечения, которое по приложению, созданному с использованием различных языков, строит графы информационных зависимостей и зависимостей по управлению, выполняет анализ потоков данных и находит разнообразные зависимости одних компонент от других. Разработанный нами продукт RescueWare Workbench, в котором решаются все перечисленные выше задачи, успешно продается на программном рынке США.

Опыт первых же использований наших инструментальных средств показал, что задача решения проблемы 2000 года существенно шире, и многие принципиально важные этапы носят скорее организационный, чем научный характер. По мере накопления практического опыта при решении конкретных задач различных фирм наше понимание проблем постепенно видоизменялось –

от проверки отдельных компонент прикладного ПО к комплексному решению проблемы 2000 года для информационной системы *в целом*.

В России не так много компаний, имеющих опыт решения проблемы 2000 года и оснащенных к тому же соответствующими инструментальными средствами. Поэтому мы одними из первых откликнулись на предложение Государственного Комитета по информатизации и связи о сотрудничестве в области решения проблемы 2000 года. Поэтому первый в России Центр Компетенции по проблеме 2000 года был создан именно на базе холдинга "ЛАНИТ".

Мы принимали участие в Парламентских Слушаниях по проблеме 2000 года в Государственной Думе РФ и конференциях, организованных Мировым банком, Американской Торгово-промышленной палатой, Британским Советом, на конференции по проблеме 2000 года для специалистов Банка России и во многих других мероприятиях.

Мы заключили договор с компаниями Microsoft и Oracle о взаимном сотрудничестве в области решения проблемы 2000 года. Согласно этому договору Центр Компетенции ЛАНИТ и представительства компании Microsoft и Oracle в России планируют предоставление совместных услуг по санации и адаптации программно-технических средств в соответствии с требованиями 2000 года, проведение совместных семинаров по проблематике 2000 года и т.п. Аналогичные переговоры ведутся с Novell, Hewlett Packard и другими компаниями.

Летом 1998 года лидер мирового рынка информационных технологий – компания Electronic Data Systems (EDS) и компания ЛАНИТ создали совместное предприятие "EDS-ЛАНИТ". Компания EDS имеет огромный опыт решения проблемы 2000 года для крупнейших заказчиков по всему миру. В связи с этим, возможности холдинга ЛАНИТ по решению проблемы 2000 года резко возросли.

2000 год: организационные проблемы

Многие ошибочно полагают, что проблеме 2000 года уделяется слишком много внимания. Например, в статье [2] проблема 2000 года пренебрежительно называется "задачей для начинающих программистов". В последнее время в прессе появилось даже несколько статей, авторы которых безаппеляционно утверждают, что проблема 2000 года не стоит того, чтобы бороться с ней.

Мы считаем, что суть решения проблемы 2000 года состоит в том, чтобы путем регулярных проверок и модификаций минимизировать возможные последствия. Очевидно, что абсолютных решений здесь нет и быть не может. Так, например, в США уже готовят юристов для разрешения судебных конфликтов, связанных с некачественным выполнением обязательств по 2000 году.

Другим, менее очевидным аспектом проблемы 2000 года является угроза потери большой части информационной инфраструктуры. Высказывая это утверждение, мы полагаем, что на решение проблемы 2000 года в критических системах деньги все-таки будут выделены (находятся же у финнов средства на проверку российских атомных станций [10], а у НАТО – на совместную с нашим Министерством Обороны программу решения проблемы 2000 года [11]). Но даже в этом случае мы рискуем потерять огромные деньги на проблеме 2000 года в менее критических системах. Это может вынудить большинство существующих предприятий и организаций, как государственных, так и частных, перейти на “запасные” варианты (использование бумажной технологии вместо электронной и т.п.), что, конечно же, приведет к огромным убыткам.

Различные подходы к решению проблемы 2000 года

Существует несколько путей решения проблемы 2000 года.

Многие организации пытаются решить эту задачу самостоятельно, силами своих сотрудников. Такой подход может оказаться успешным только в том случае, если программные продукты, используемые фирмой, могут быть целиком исправлены собственными специалистами, при условии, что программисты, создававшие их, все еще работают в этой организации. В реальной практике зачастую не выполняются оба этих условия.

Если в вашей фирме используется значительное число программ, то удачным решением может стать покупка программного продукта, автоматизирующего решение проблемы 2000 года. Однако, даже правильный выбор продукта может оказаться очень сложной задачей. К сожалению, в таком случае часто приобретается продукт "для галочки", то есть исключительно для того, чтобы можно было отрапортовать начальству. Реального увеличения производительности в таких случаях не происходит.

Наконец, одним из самых надежных решений является заключение соглашения с консалтинговой фирмой, специализирующейся на решении данной проблемы. Основным достоинством этого подхода является то, что над системами, требующими исправления, будет работать большее число сотрудников высокой квалификации, чем можно себе позволить в рамках своей фирмы.

Общий план решения проблемы 2000 года

Мы понимаем решение проблемы 2000 года как последовательное выполнение следующих шагов.

1. Рассмотрение информационной системы в целом и проведение инвентаризации ее компонент:
    - ЭВМ, маршрутизаторов, сетевых карт и других аппаратных средств;

    - ОС, сетевого ПО, протоколов стыков, серверов баз данных и другого системного ПО;
    - Прикладное ПО и конкретные данные в БД.
2. Анализ используемых в информационной системе аппаратных средств на базе информации, предоставляемой ведущими фирмами-изготовителями компьютерной техники, такими как IBM, Dell, Compaq, Hewlett Packard и так далее, и сетевых аппаратных средств, такими как 3Com, Cisco Systems и так далее. Составление предложений по замене компьютеров либо их отдельных блоков.
3. Анализ версий используемого системного ПО на базе информации, предоставляемой ведущими фирмами-разработчиками системного ПО и СУБД, такими как Microsoft, Novell, IBM, Sun Microsystems, Oracle, Sybase и так далее. Выработка предложений по их централизованной модификации.
4. Проведение анализа и модификации прикладного ПО.
5. Комплексная отладка и тестирование модернизированной информационной системы.

Процесс решения проблемы 2000 года может быть итеративным, то есть на поздних этапах решения проблемы (например, на этапе комплексной отладки) может возникнуть необходимость возврата к уже выполненным шагам и их повторному выполнению, постепенно увеличивая накопленные знания о системе и учитывая ее вновь открытые особенности.

Инвентаризация

Начальным этапом решения проблемы 2000 года для любой информационной системы является ее инвентаризация, то есть, комплексное обследование всех компонент системы и определение их готовности к 2000 году.

Инвентаризация компонент большой корпоративной информационной системы – это серьезная и объемная работа, на которую может приходиться до половины всех усилий, затрачиваемых на решение проблемы в целом. При этом основной упор должен быть сделан на организационные, а не на технические меры – рассылку анкет, получение полной и своевременной информации об используемых в системе средствах, версиях системного ПО, типах используемого оборудования и так далее. Как показывает наш опыт работы, это практически никогда не удается сделать с первого раза – всегда остаются неразрешенные внешние ссылки на неописанные компоненты системы, информацию о которых приходится "вытаскивать" дополнительно. Еще одна, чисто организационная проблема – обеспечение понимания важности процесса со стороны конечных пользователей системы, чтобы избежать отношения к спускаемым "сверху" инвентаризационным анкетам как к бесполезным бюрократическим требованиям.

Инвентаризация входящего в состав информационной системы прикладного ПО создает свои особые проблемы. Возможна ситуация, когда для каких-то частей

ПО, используемых в системе, сопровождение вообще не ведется, вплоть до утраты части исходных текстов программ или их несоответствия исполняемому коду. В принципе, такая ситуация может оказаться критической для решения проблемы 2000 года в системе в целом. Однако наш опыт работы показывает, что если доля таких "бесхозных" программ невелика (то есть составляет не более 10%), то проблема 2000 года для этих компонент может решаться в рабочем порядке, не оказывая серьезного влияния на систему в целом.

Модернизация аппаратных средств

Многие компании предлагают решить проблему 2000 года для аппаратуры путем полной замены компьютерного парка. Это действенное решение, но может оказаться разорительным для заказчика. Основываясь на анализе существующих аппаратных средств, мы считаем более целесообразным решение, связанное с частичной модернизацией (upgrade) компьютеров.

Во многих компьютерах, особенно изготовленных недавно, проблемы 2000 года не существует или для ее решения достаточно произвести перезапуск с ручной установкой даты (например, 1 января 2000 года или какие-то другие установки).

Если необходима модернизация аппаратуры, она, как правило, заключается в обновлении BIOS и может быть разделена на следующие категории:

- Загрузка новой версии BIOS с помощью специального программного обеспечения. Такая возможность существует для компьютеров с Flash-BIOS. Большинство компьютеров выпуска позднее 1996 года (как, впрочем, и ряд более ранних моделей) обладают такой возможностью. Стоимость необходимого оборудования в таком случае равна нулю.
- Обновление BIOS при помощи специального оборудования, например программатора, либо замена на новую микросхему. Стоимость необходимого оборудования имеет порядок единиц долларов (стоимость микросхемы ПЗУ).

Списки компьютеров с указанием их совместимости с проблемой 2000 года обычно публикуются на официальных WWW-серверах фирм-производителей, напримерhttp://www.ibm.com/ – сервер фирмы IBM, http://www.compaq.com/ – сервер фирмы Compaq. Отметим, что для компьютеров, не попадающих в категорию brand-name, необходимо изучать информацию на Web-серверах изготовителей материнской платы.

В любом случае выводы о необходимости той или иной модернизации могут быть сделаны только после тщательного тестирования и анализа оборудования, используемого программного обеспечения и области использования системы.

Особняком стоит задача решения проблемы 2000 года для телекоммуникационного и прочего оборудования. Иногда удается непосредственным тестированием доказать наличие у такого оборудования

несовместимости с 2000 годом, но, как известно, никаким набором тестов невозможно доказать *отсутствие* ошибки. Поэтому при решения проблемы 2000 года для такого оборудования необходимо работать в тесном контакте с фирмой-изготовителем.

Модернизация системного программного обеспечения

Для большинства фирм-производителей системного ПО ранние версии их продуктов не являются готовыми к 2000 году или являются частично готовыми. Однако фирмы-производители предлагают варианты обновлений своих ОС или типовых приложений, либо замены их более новыми на льготных условиях. Например, наиболее распространенная операционная система Windows 95 не обеспечивает корректной работы с датами. Недавно фирма Microsoft выпустила ОС Windows 98, в которой одним из основных достоинств объявлено исправление проблемы 2000 года и рекомендована цена обновления 90$. В некоторых случаях фирмы-изготовители свободно распространяют так называемые "заплатки" (patches), с помощью которых решается проблема 2000 года (приведем в качестве примера Windows NT 4.0), в других случаях требуется просто замена системы на более позднюю версию, в которой проблема 2000 года решена.

Особенностью России по сравнению с остальными странами является повсеместное использование нелицензированных копий операционных систем и массовых приложений. Большинство таких продуктов существенно устарели и обычно подлежат замене на более новые версии для решения проблемы 2000 года. Единственный выход из данной ситуации состоит в организации централизованных закупок с льготными условиями и переговоров об организации легализации. Холдинг ЛАНИТ ведет подобные переговоры с большим количеством фирм-изготовителей системного ПО.

Для решения проблемы 2000 года в части аппаратуры и системного ПО Центр Компетенции на базе ЛАНИТ использует базу знаний, накопленную компанией EDS. На основе этой базы знаний построен продукт Vendor 2000, позволяющий быстро и эффективно находить информацию более чем о 20 000 различных программных и аппаратных продуктов. Данные базы знаний обновляются ежедневно, что позволяет постоянно получать оперативную и точную информацию.

Решение проблемы 2000 года для прикладного ПО

Решение проблемы 2000 года для прикладного ПО является самой сложной и наукоемкой частью всего процесса в целом. Хотя специфичные для предприятия прикладные программы могут составлять всего около 10% всех компонент корпоративной информационной системы, решение проблемы 2000 года для них может потребовать до 90% всех усилий.

Дело в том, что каждая прикладная система является сугубо индивидуальной, и к ней неприменимы стандартные методы, используемые для аппаратуры и системного ПО. Для каждой системы приходится проводить анализ ее исходных текстов с целью поиска "подозрительных" мест, а затем согласованно исправлять все некорректные с точки зрения проблемы 2000 года фрагменты алгоритмов.

Программы состоят, как правило, из большого количества модулей, взаимодействующих друг с другом через файлы, таблицы баз данных или экранные формы, и поэтому для корректного решения проблемы 2000 года нужен достаточно глубокий анализ всего приложения в целом, учитывающий возможную передачу данных (в том числе, и содержащих дату) между модулями. Поэтому средства автоматизации решения проблемы 2000 года могут быть четко разделены на две категории: простые сканеры, позволяющие быстро проводить первичный анализ, и большие системы, выполняющие глубокий анализ приложения.

Простейшие средства для решения проблемы 2000 года

Начальный этап решения проблемы 2000 года для прикладного ПО состоит в анализе приложений, составляющих программную часть информационной системы, и оценки объема работ, которые должны быть выполнены для успешного решения проблемы 2000 года. В этой работе существенную помощь могут оказать программы-сканеры, которые с помощью набора шаблонов и эвристик осуществляют быстрый поиск участков программ, подозрительных на несовместимость с 2000 годом.

Типичным примером такой программы-сканера является разработанная специалистами ЛАНИТ-ТЕРКОМ система ***"Сапер 2000"***. Основным достоинством данной системы является легкость настройки на конкретный язык программирования, что позволяет проводить с ее помощью анализ программ, написанных с использованием практически любого регулярного языка. Другим важным преимуществом системы "Сапер 2000" является быстрота анализа, поскольку производится только поверхностный анализ приложения, без учета особенностей языка приложения и межмодульных взаимодействий. Система имеет интуитивно понятный интерфейс, позволяющий быстро начать работу любому пользователю. Стоит отметить дополнительную особенность "Сапера 2000" – возможность генерировать новые исходные тексты программ с комментариями, отражающими результаты анализа и необходимые исправления. Комментарии вставляются непосредственно перед вхождением идентификатора, требующего исправлений, и включают метод исправления переменной, который нужно применить, а также комментарий, введенный пользователем для этой переменной. Эти исходные тексты можно использовать непосредственно в процессе дальнейшего исправления приложения. После анализа приложения можно получить отчеты о подозрительных переменных, о переменных, требующих исправлений, а также отчеты по составу проекта.

К сожалению, многие компании, специализирующиеся на выпуске программного обеспечения по проблеме 2000 года, позиционируют подобные программы-сканеры как законченное средство для решения проблемы. На самом деле, подобные средства пригодны только для начального этапа оценки сложности. Именно поэтому мы всегда рекомендуем при выборе средств автоматизации решения проблемы 2000 года прибегать к услугам консалтинговых фирм.

Продвинутые средства для решения проблемы 2000 года

Глубокий анализ прикладных программных комплексов является весьма сложной задачей, которую практически невозможно решить "с нуля", не имея серьезных заделов в области анализа программных систем. Поэтому неудивительно, что основные позиции в этом секторе рынка прочно занимают разработки компаний, работающих в области автоматизации реинжиниринга, т.е., переноса больших прикладных комплексов с устаревших платформ в новые технологические среды. Продукты, предлагаемые такими фирмами, используют для решения проблемы 2000 года весь объем наработок по регистрации и учету компонент приложений, их синтаксическому анализу, отслеживанию разнообразных связей и т.п. Вся информация об анализируемом приложении собирается в общей объектно-ориентированной базе данных (репозитории), наличие которого может считаться отличительным признаком этого класса продуктов. Средства автоматизации решения проблемы 2000 года, как правило, оформляются в виде специализированных компонент, надстроенных над стандартными средствами анализа.

Одним из продуктов такого класса является система реинжиниринга ***"Сталкер"***, являющаяся русифицированной версией программного пакета RescueWare и разработанная специалистами ЛАНИТ-ТЕРКОМ. Система предназначена для автоматизированного анализа программных комплексов и перевода их с устаревших платформ, в первую очередь типа mainframe, на современные платформы и технологические средства на базе Client/Server или Internet/Intranet под управлением операционной системы Windows 95/98/NT. Система включает в себя как автоматизированные средства перевода программной логики, так и средства реорганизации пользовательского интерфейса и компонент работы с базами данных. Удобный графический интерфейс, хранение всей информации в общей объектно-ориентированной БД (репозитории) делает работу пользователя с системой удобной и интуитивно ясной.

Система поддерживает большое количество входных и выходых языков, причем ее архитектура заранее рассчитана на возможность расширения. Включение нового входного (или выходного) языка сводится к написанию соответствующего синтаксического анализатора или просмотра генерации. При этом все внутренние структуры данных, в которых представляется информация об исходной программе, являются универсальными и языково-независимыми, так что все алгоритмы глубокого анализа программ автоматически оказываются

применимыми и для приложений на новом языке. В настоящее время в качестве выходных языков выступают C++, Java и Visual Basic, а в качестве входных (с разной степенью завершенности) – Cobol, PL/I, Fortran, Clipper, FoxPro.

Понятно, что процесс анализа и модификации старых программ не может быть полностью автоматическим – всегда останутся неоднозначные ситуации, решение в которых может быть принято только человеком. Средства, предоставляемые системой "Сталкер", могут лишь в большей или меньшей степени автоматизировать этот процесс.

Средства решения проблемы 2000 года, поддерживаемые системой "Сталкер"

Методы исправления программ для решения проблемы 2000 года сводятся к решению двух задач: необходимо найти переменные, содержащие дату, и исправить программу так, чтобы она оперировала с правильным форматом этой даты.

Различным способам проведения подобных преобразований посвящено множество статей, например, краткое описание большинства таких методов содержится в статье [5]. Существуют также некоторые нетривиальные алгоритмы, основанные на изменении системы времяисчисления, призванные сохранить неизменным 6-разрядный формат даты (имеется в виду формат YYMMDD). Один из таких подходов опубликован в [6].

В последнее время появились подобные статьи и на русском языке [7], но, к сожалению, методы исправления программ все еще недостаточно полно освещены.

На первый взгляд, эти задачи кажутся несложными – достаточно внимательно просмотреть модуль, оперирующий с датами, и внести необходимые изменения в формат переменных. Но на практике приходится сталкиваться с системами, состоящими из большого количества модулей, использующих одни и те же данные. В результате граф межмодульных взаимодействий может быть настолько сложным и запутанным, что человек будет не в состоянии проследить миграцию тех или иных данных в программе. И даже когда уже обнаружены все переменные, в том или ином виде содержащие дату, а также все места их использования, процесс исправления приложения без использования специализированных средств автоматизации займет большое количество времени без какой-либо гарантии его корректности.

Поиск "подозрительных" переменных

В процессе синтаксического анализа программного комплекса создается база данных, в которой содержится информация обо всех переменных, операторах, целочисленных и строковых константах и т.п. По созданной базе производится поиск "подозрительных" переменных, который включает в себя три этапа:

начальное сканирование, локальный анализ и глобальный анализ с учетом межмодульных взаимодействий.

Начальное сканирование

На этом этапе производится поиск "подозрительных" переменных в пределах одного модуля, при этом существенная роль отводится пользователю. Очень часто переменные, содержащие дату, имеют характерное название: "year", "YY", "date", "god" и т.п. Такие переменные несложно обнаружить и выделить с помощью набора шаблонов. Шаблоны могут задавать как признаки "подозрительности" переменной, так и признаки, которыми такие переменные обладать не должны. Важен также тот факт, что переменные, содержащие дату, широко используются только в ограниченном наборе конструкций языка, например, в операциях сравнения и операциях ввода-вывода. Применение основанных на данном факте эвристик позволяет сократить и конкретизировать набор переменных, найденных по шаблону. Пользователю должна быть предоставлена возможность полностью контролировать процесс начального поиска, менять его критерии, задавать свои шаблоны. Кроме того, пользователь должен иметь возможность подтвердить или опровергнуть "подозрительность" каждой из найденных переменных, а также пополнить список таких переменных, основываясь на собственных заключениях.

Целью этого этапа является создание "основания", т.е. некого набора переменных, гарантированно содержащих дату. Этот набор будет использован на последующих этапах для обнаружения полного набора переменных и точек в программе, страдающих от проблемы 2000 года. Таким образом, от результатов начального сканирования зависит эффективность работы более сложных алгоритмов поиска дат.

Локальный анализ

На этапе локального анализа происходит расширение списка переменных, найденных на этапе начального сканирования. Анализируя поток данных программы, алгоритм разбивает множество всех переменные на *классы эквивалентности*. Две переменные попадают в один класс в том случае, если они одновременно участвуют в какой-либо операции, для которой критично совпадение их типов (форматов). По окончании работы алгоритма мы имеем набор классов переменных, причем про каждый класс можно строго сказать, что либо все переменные этого класса в то или иное время содержат дату, либо ни одна из них не содержит. Теперь для каждой переменной, найденной на этапе начального сканирования, мы имеем целый класс "подозрительных" переменных.

Пример образования классов эквивалентности:

```
/* Описание переменных */
char Date1[6];
char Birthday1[6];
```

```
char Date2[6];
char Birthday2[6];

/* Тело программы */
if (Date1>Birthday1)
printf("Error!");
else
if (Date2>"010160")
Birthday2 = Date2;
```

После анализа текста данного примера образуется 2 класса эквивалентности `(Date1,Birthday1); (Date2,Birthday2).`

Применяя такой подход, мы можем гарантировать то, что в пределах одного модуля будут найдены *все* переменные, содержащие дату, и не будет найдено лишних, так как мы проводим полный анализ потока данных в пределах модуля.

Глобальный анализ приложения с учетом межмодульных взаимодействий

До сих пор мы говорили только о поиске переменных в пределах одного модуля, но на практике информация имеет тенденцию "мигрировать" из одной компоненты приложения в другую, поэтому для полного отслеживания передачи данных в приложении необходимо иметь средства анализа потока данных не только для каждого модуля по отдельности, но и для программного комплекса в целом. На основе информации, собранной на этапе синтаксического анализа, производится глобальный анализ с целью установления эквивалентности между переменными из разных модулей. Отслеживается миграция данных через различные типы файлов, таблицы баз данных, экранные формы и параметры, передаваемые при вызовах программ; каждой такой сущности сопоставляется некоторая абстрактная переменная, которую мы называем глобальной. Добавляя эти знания к ранее собранной информации о связях между переменными в каждом из модулей программного комплекса и оперируя с глобальными переменными так же, как и с обычными (локальными), мы строим общую таблицу транзитивного замыкания по отношению эквивалентности с точки зрения "подозрительности" на содержание даты.

Пример:

first.c

```
void Retrieve_Date()
{
char *SQL = "Select MM0, DD0, YY0 Into Month, Day, Year From DB-Table";
ExecSQL(DataBase, SQL);
};
```

second.c

```
FILE *Map1 = fopen("Map1.scr", "w");
fprintf(Map1, "%s", NewYear);
fclose(Map1);
```

third.c

```
FILE *Map1 = fopen("Map1.scr", "r");
fscanf(Map1, "%s", SomeYear);
fclose(Map1);

void Save_Date()
{
char *SQL = "Insert Into DB-Table (MM0, DD0, YY0) Values (SomeMonth, SomeDay,
SomeYear)";
ExecSQL(DataBase, SQL);
};
```

После анализа данного примера глобальная переменная, соответствующая файлу `Map1`, и глобальная переменная, соответствующая колонке таблицы `DB-Table.YY0`, попадут в один класс эквивалентности. Таким образом, появляется возможность для любой локальной переменной каждого из этих модулей автоматически найти связанные с ней по формату переменные и константы не только в этом, но и в других модулях.

Дополнительные усовершенствования алгоритма поиска

По результатам проведенных нами исследований было установлено, что довольно большой процент информации передается не напрямую из переменной в переменную, а может временно являться составной частью какой-либо глобальной структуры. То есть информация сначала является распределенной по переменным, затем собирается вся вместе, как-либо трансформируется, а затем уже снова распределяется по переменным, причем зачастую совсем другим. При этом чрезвычайно сложно установить взаимосвязь между исходным и окончательным набором переменных. Поэтому важно иметь возможность оперировать не только переменными, но и их частями, в идеале нужно отдельно отслеживать каждый байт информации. Проиллюстрируем это на примере.

Пример:

```
/* Описание переменных */
struct DateType
{
char MM[2];
char DD[2];
char YY[2];
};
DateType Date1;
char Birthday1[6];
char Date2[6];
char Birthday2[6];

/* Участок программы */
if (Date1.YY>substr(Birthday1, 5, 2))
printf("Error!");
else
if (Date2>"010160")
Birthday2=Date2;
```

После анализа данного примера образуется 2 класса эквивалентности `(YY,Birthday1(5:2)); (Date2,Birthday2)`. Если же анализировать этот пример, не имея возможности работать с участками переменных, первый класс эквивалентности создан не будет.

Аналогично отслеживаются эквивалентности участков переменных при межмодульных взаимодействиях.

Реализация различных методов изменения исходного текста программ

Процесс внесения изменений в исходную программу с целью обеспечения корректности с точки зрения проблемы 2000 года называется *исправлением программы (Remediation)*. В настоящее время наиболее распространены следующие методы исправления:

Расширение переменных (Expansion)

Расширение переменных является простейшим методом исправления программы. Переменная, к которой был применен данный метод, преобразуется следующим образом:

В описании переменной к ее представлению добавляются два дополнительных поля (символа). Например, если у нас была переменная X, описанная следующим образом:

```
char X[6];
```

то после применения метода ее описание будет иметь следующий вид:

```
char X[8];
```

В случае, когда метод применяется к структуре, т.е. переменной, не имеющей описания формата, то либо в эту структуру будет добавлено фиктивное поле, в котором будет содержаться информация о столетии, либо будет указано поле структуры, содержащее описание формата , которое должно подвергнуться расширению.

Пример:

```
struct CStruct
{
char Field1[4];
char Field2[2];
};

CStruct SOME_STRUCTURE;
```

После применения метода без указания поля:

```
struct CStruct
{
```

```
char YY_Field[2];
char Field1[4];
char Field2[2];
};

CStruct SOME_STRUCTURE;
```

После применения метода с указанием поля FIELD2:

```
struct CStruct
{
char Field1[4];
char Field2[4];
};

CStruct SOME_STRUCTURE;
```

В местах, где переменная участвует в операциях сравнения, необходимо проверить все выражение на наличие констант, с которыми данная переменная может сравниваться. Если такие константы присутствуют, с ними необходимо выполнить то же преобразование, что и с переменной, иначе операция сравнения будет неверной.

Пример:

```
if (X > "770924") printf("Greater!");
```

После применения метода:

```
if (X > "19770924") printf("Greater!");
```

В случаях, подобных этому, всегда можно предполагать, что для преобразования константы к верному виду достаточно добавить к ней слева "19", т.к. программа создавалась до 2000 года и не была рассчитана на работу с датами следующего столетия, поэтому и все константы в ней относятся к XX веку.

Реализация первой части метода (расширение представления) чрезвычайно проста: достаточно последовательно пройти по всем описаниям переменных, которые должны быть подвергнуты расширению, и расширить их на 2 символа.

Реализация второй части (изменение констант) также не представляет особых проблем. При анализе дерева разбора программы просматриваются все операции сравнения и если одним из операндов является переменная, к которой применяется расширение, то все константы во втором операнде, которые совпадают по размеру с переменной, преобразуются.

Существенным недостатком описанного метода является то, что при изменении длины представления все операторы, для которых длина переменной играет существенную роль (в первую очередь, операторы вывода), будут неправильно или не вполне правильно работать с обновленными переменными. Например, если мы имеем некоторую экранную форму, которая была создана с расчетом на то, что длина выводимой переменной X равна 6 символам, то после применения к X метода расширение ее длина увеличится до 8 символов и при выводе ее на экран терминала часть значения может быть не видна, или наоборот, дополнительные символы могут закрыть собой какую-либо нужную информацию. Больше всего от изменения длины могут пострадать операторы работы с файлами. Несовпадение длины записи в файле с длиной структуры, в которую эта запись будет считываться, может привести к непредсказуемым последствиям. Таким образом, мы сталкиваемся с необходимостью осуществлять преобразование огромного количества информации, существующей в виде файлов и разнообразных баз данных.

Перечисленные проблемы сильно ограничивают область применения метода расширения переменных.

Метод с Логическим Окном (Windowing)

Более сложным является метод **с** *логическим окном*, лишенный многих недостатков метода расширения переменных. Метод состоит из следующих частей:

Для каждой переменной, к которой применяется метод, заводится так называемая *переменная для использования в окне*. Это глобальная переменная, получающаяся из оригинальной добавлением к ее представлению двух дополнительных полей (символов). Для краткости, в дальнейшем будем такие переменные называть W-переменными.

Пример:

```
struct CStruct
{
char X[6];
};

CStruct SOME_STRUCTURE;
```

После применения метода с логическим окном к переменной X:

```
struct CStruct
{
char X[6];
};

char W_X_SOME_STRUCTURE[8];
CStruct SOME_STRUCTURE;
```

В тех местах исходной программы, где переменная участвует в операциях сравнения, вместо оригинальной переменной будет подставлена соответствующая ей W-переменная, имеющая расширенный формат, чем обеспечивается корректность сравнения с точки зрения проблемы 2000 года. Чтобы сравнение было корректным с точки зрения логики программы, необходимо, чтобы на момент исполнения операции сравнения обе переменные (оригинальная и W-переменная), содержали одну и ту же дату. Это достигается за счет того, что непосредственно перед оператором, в котором встретилось сравнение, вставляется участок кода, "синхронизирующий" значения оригинальной и W-переменной. В процессе синхронизации необходимо по дате, содержащейся в исходной переменной, решить, относится эта дата к XX или XXI веку. Для этого число, соответствующее году в оригинальной переменной (состоящее из двух цифр), сравнивается с некоторым фиксированным значением, называемым *граничной датой* (Pivot Date). Если наше число больше этого значения, то считается, что дата принадлежит XX веку, и четырехсимвольное представление года в W-переменной будет начинаться с "19", иначе считаем, что это – дата XXI века и представление года в W-переменной будет начинаться с "20". Как и в случае с методом расширения переменных, необходимо также выполнить преобразование констант.

Пример:

```
if (SOME_STRUCTURE.X>"770924") printf("Greater!");
```

После применения метода с логическим окном к переменной X:

```
if (substr(SOME_STRUCTURE.X, 1, 2) > "50")
substr(W_X_SOME_STRUCTURE,1, 2)="19";
else
substr(W_X_SOME_STRUCTURE, 1, 2)="20";
substr(W_X_SOME_STRUCTURE, 3, 6)=SOME_STRUCTURE.X;
if (W_X_SOME_STRUCTURE >"19770924") printf("Greater!");
```

В данном примере в роли граничной даты выступает число 50 в первой строчке текста после преобразования. Если в качестве граничной даты выступает какая-либо константа, то говорят, что применяется метод *с фиксированным логическим окном* (with Fixed Window). Если же значение граничной даты хранится в какой-либо переменной, и все сравнения ведутся с этой переменной, то говорят, что применяется метод *со скользящим логическим окном* (with Sliding Window). Заметим, что для проведения сравнения с граничной датой нам необходимо знать некоторую дополнительную информацию о формате оригинальной переменной, а именно, точное положение тех двух цифр, которые представляют год. Предполагается, что данная информация будет предоставлена пользователем.

Для сохранения синтаксической правильности программы при внесении изменения в операцию сравнения участок кода, ответственный за

синхронизацию, должен быть вынесен за границу оператора, в котором встретилась операция. Наиболее просто это реализуется следующим образом. При обработке очередного оператора программы, ссылка на узел дерева разбора, представляющего этот оператор, запоминается. Назовем эту ссылку указателем последнего оператора. Если оператор, в котором встретилась операция, требующая изменения текста программы, находится на том же уровне вложенности, что и оператор, на который у нас есть ссылка, то код синхронизации вставляется непосредственно между ними и указатель последнего оператора перемещается на вставленный код. Когда мы при анализе дерева разбора входим в очередной блок вложенности, текущее значение указателя последнего оператора помещается на стек, а ему присваивается адрес конструкции, служащей началом блока. При выходе из блока, указатель последнего оператора снимается со стека.

Таким образом, все операции, которые могут оказаться некорректными с точки зрения проблемы 2000 года, проводятся над W-переменными, что гарантирует нам их правильность. С другой стороны, мы избегаем проблем, которые могут возникнуть из-за изменения размера представления переменной, ибо все остальные операции, включая операции ввода-вывода, производятся над оригинальными переменными неизмененной длины. При этом также отпадает необходимость делать преобразования данных, хранящихся в файлах.

Главным недостатком метода является то, что мы получаем только временное решение проблемы, особенно при методе с фиксированным окном, т.к. проблема заново возникнет при приближении года, используемого как граничная дата.

Метод минимизации количества логических окон (Global Windowing)

Еще одним недостатком метода с логическим окном является то, что тексты программ могут довольно сильно увеличиться за счет участков синхронизации, особенно если переменные, к которым был применен метод, часто используются в программе. Для решения последней проблемы был придуман метод, носящий название Global Windowing. Он предусматривает глубокий анализ потоков данных в программе с целью минимизировать количество вставок для синхронизации дат. После проведения такого анализа мы можем выявить те места в программе, где значения оригинальной и W-переменной еще не успели рассинхронизироваться. Это позволяет существенно сократить количество вставок. К сожалению, на реальных приложениях, которые могут занимать сотни тысяч строк кода, такой анализ может занять очень много времени и улучшения, полученные с его помощью, не оправдают таких временных затрат. Поэтому практическая польза метода *минимизации количества логических окон* представляется сомнительной.

Методы кодирования и упаковки

Суть методов *кодирования* и *упаковки* заключается в том, что для хранения значения года используются те же двухсимвольные переменные, что и ранее, но данные в них записываются не в явном, а в упакованном виде. Как известно, с помощью двух байтов можно представить числа в диапазоне от 0 до 65535. Поэтому если использовать не явное (от "00" до "99") представление, а весь доступный диапазон значений, то правильная работа программы нам будет гарантированна до 65535-го года.

При применении этого метода необходимо изменить все константы, представляющие дату, которые присутствуют в приложении. Для того, чтобы сделать кодирование дат прозрачным с точки зрения пользователя, во всех участках программы, в которых осуществляется общение с пользователем и используются даты, необходимо вставить код, осуществляющий кодирование и декодирование дат.

Пример:

```
X = "77";
printf("%s", X);
```

После применения метода кодирования:

```
X[0] = 0x'07'; // 07B9 является шестнадцатеричным
X[1] = 0x'B9'; // представлением числа 1997

// декодирование даты для вывода на экран
unsigned int TempVar = ((int)X[0] << 8) + (int)X[1];
char TempDate[2];
TempDate[0] = '0' + (char)((TempVar / 10) % 10);
TempDate[1] = '0' + (char)(TempVar % 10);
printf("%s", TempDate);
```

Как и в случае с методом расширения переменных, существенным недостатком является необходимость преобразования накопленных баз данных, чтобы обеспечить нормальную работу с ними. Кроме того, использование упакованного формата понижает общую читабельность программы и возможность ее дальнейшего сопровождения.

Исправление текста при работе с полями переменных

Как говорилось ранее, если минимальной рассматриваемой единицей при анализе считать переменную, то корректно решить проблему 2000 года в общем случае невозможно. Поэтому вводилось понятие поля переменной и анализ проводился над полями переменных. Соответственно, можно изменить и методы исправления программы, чтобы по результатам такого анализа выполнить преобразование исходных текстов. Применяемые при этом методы аналогичны описанным выше, только единицей исправления является не переменная, а какое-либо поле этой переменной.

Все описанные методы исправления программ реализованы в системе "Сталкер" и накоплен большой опыт их применения на практике. Каждый из методов исправления текста применяется системой полностью автоматически, измененные участки текста помечаются комментариями с объяснениями внесенных изменений. Так как в процессе исправления программы учитывается вся информация о константах и частях переменных, а также межмодульных отношениях, то результирующая программа получается корректной и часто требует лишь небольшой косметической правки.

Перспективные пути решения проблемы 2000 года для прикладного программного обеспечения

В мировой практике встречается задача восстановления знаний о программе по ее исходному тексту (reverse engineering). Такие задачи порою носят самостоятельный характер и решаются в целях облегчения сопровождаемости программ. Как мы уже видели, для решения проблемы 2000 года требуется достаточно глубокий анализ существующего приложения, поэтому после решения проблемы 2000 года должно появиться утерянное с годами понимание программного комплекса, например, знание информационных потоков и межмодульного взаимодействия. Таким образом, при грамотном решении проблемы 2000 года у исправляемой системы все-таки появляется новое качество.

Мы полагаем, что еще более дальновидное решение – не останавливаться на достигнутом и выполнить не только решение проблемы 2000 года, но и реинжиниринг программного комплекса. Дело в том, что программы устаревают чрезвычайно быстро – за 10 лет успевают смениться не только аппаратные платформы, но и идеологии программирования. Поэтому переписывание полезных программ для удовлетворения современных потребностей неизбежно. По нашим оценкам, глубокий анализ для решения проблемы 2000 года занимает от четверти до трети общего объема работ по реинжинирингу.

Краткое освещение методологии реинжиниринга программного обеспечения дано в [1].

Проблема 2000 года в сети

Любая серьезная информационная система массово использует преимущества корпоративной вычислительной сети. Но с точки зрения решения проблемы 2000 года это создает целый класс новых проблем. Все начинается с того, что в сети обычно используется техника разных фирм-производителей, и даже если каждая из этих фирм по отдельности гарантирует работу своего оборудования в новом тысячелетии, это не означает, что вся сеть будет работать в комплексе [9]. Кроме того, при работе в сети зачастую не очевидно, из каких источников то или иное устройство получает информацию о системной дате.

Для обнаружения проблем, связанных с взаимодействием различных устройств в сети, рекомендуется выполнить следующие процедуры.

- Проверить системы обмена сообщениями. Для этого необходимо изучить все административные процедуры и выяснить, в каких операциях используются даты, а также исследовать системы электронной почты и проверить их системы автоматической регистрации.
- Провести ревизию используемых маршрутизаторов и коммутаторов и проанализировать их совместимость с 2000 годом. Необходимо, используя данные фирм-производителей, попытаться найти несоответствия в обработке даты на маршрутизаторах и сетевых серверах времени, проверить, как генерируются предупредительные сигналы в маршрутизаторах и коммутаторах.
- Если информационная система построена на основе сетевой операционной системы, то необходимо проверить следующие пункты: отсутствие влияния неправильной даты на файловую систему сети, корректная обработка даты в системах защиты данных, программы резервного копирования. Дополнительно необходимо изучить взаимодействие аппаратного обеспечения персональных компьютеров, сетевой ОС, ОС на клиентских станциях.

Обнаружение и устранение проблемы 2000 года в большой вычислительной сети – чрезвычайно сложная работа и должна быть выполнена силами организации, имеющей опыт в решении такого рода задач. Поэтому при возникновении подозрений о наличии проблемы 2000 года в сети мы рекомендуем обращаться к услугам системного интегратора.

Тестирование и комплексная отладка

К началу работы на данном этапе модификация аппаратных средств, операционных систем и прикладного ПО для достижения ими совместимости с 2000 годом должна быть уже завершена.

При тестировании модифицированного комплекса проверяется соответствие модифицированной системы исходной и дополнительно проверяются следующие критерии:

- правильное отображение дат от настоящего времени и до 2034 года (в 2035 году у персональных компьютеров переполнится счетчик времени);
- корректный переход через границу 2000 года;
- корректность алгоритма вычисления високосного года;
- корректность вычислений дней недели для всех дат от 1980 до 2034 года;
- правильное вычисление временных диапазонов в промежутке от 1980 до 2034 года;
- правильность сортировки временной информации в промежутке от 1980 до 2034 года;
- корректность обработки отсутствия временной информации.

Для повышения эффективности работы желательно использование средств автоматизации процесса тестирования. В Центре Компетенции ЛАНИТ для этого используется QACenter фирмы Compuware.

Наибольшие трудности вызывает комплексная отладка системы. Эта задача особенно трудна и ответственна для больших информационных систем. К сожалению, даже после выполнения всех описанных выше проверок гарантировать корректность работы невозможно. Известны случаи, когда программные продукты одной фирмы, корректные с точки зрения проблемы 2000 года по отдельности, создают проблемы при совместной работе. Например, различные программы могут воспринимать данные в разных форматах – американском и европейском. В этом случае при передаче данных из одной программы в другую могут возникнуть несоответствия. Задача осложняется тем, что большинство компаний, производящих программное или аппаратное обеспечение, не считают необходимым разглашение подобной информации и при этом не предоставляют никаких гарантий, связанных с 2000 годом.

Поэтому для обнаружения ошибок такого рода необходимо проведение всеобъемлющего тестирования системы с особым упором на вопросы межмодульного взаимодействия, пересылки данных по сети, общения с пользователем системы, – то есть, так называемым "местам стыка". Их надо внимательно анализировать на предмет возможных пересылок дат.

Это кропотливая и непростая работа, требующая досконального знания всех особенностей анализируемой системы. На практике она под силу только крупным системным интеграторам.

Внедрение

По окончании полного цикла решения проблемы 2000 года систему можно вводить в эксплуатацию. Обычно для этого используется копия реальных данных, и какое-то время модифицированная и старая системы работают параллельно. Если в течение некоторого времени новая система работает без сбоев, то ее переводят в "боевой" режим.

На данный момент на российском рынке представлено очень мало примеров систем, в которых был бы проделан полный цикл решения проблемы 2000 года вплоть до внедрения. Это вызвано различными причинами. Во-первых, таких систем немного. Во-вторых, корпорации, решившие проблему 2000 года в своих системах, проявляют нежелание разглашать найденные ошибки и трудности их преодоления.

Одним из наиболее подробно освещенных с технической точки зрения случаев стало решение проблемы 2000 года для системы резервирования мест и продажи железнодорожных билетов "Экспресс-2" [8]. Эта задача включала в себя проведение модификации исходных текстов комплекса, состоящего из более полумиллиона строк на макроассемблере (при этом аппаратная часть

системы не претерпела изменений). Окончательное внедрение этой системы было проведено в феврале 1999 года.

Сравнение некоторых инструментальных средств решения проблемы 2000 года

Таблица 1. Сравнение некоторых известных средств решения проблемы 2000 года.

	Система "Сталкер"	PLATINUM TransCentury Office	VIASOFT OnMark 2000	WRQ Express 2000	CTSI Target2K
Тестирование аппаратуры		√	√	√	
Анализ системного ПО		√		√	
Возможности инвентаризации	√	√	√	√	√
Синтаксический анализ	√		√		
Расширяемость множества входных языков	√		√		√
Генерация отчетов	√	√	√		√
Анализ межмодульных взаимодействий	√				
Автоматизированное преобразование исходных текстов	√				
Тестирование преобразованного комплекса		√	√*		
Возможности реинжиниринга	√				

Примечания: * – тестирование производится отдельным средством (VIA/AutoTest).

Из таблицы видно, что с помощью системы "Сталкер" можно решить большинство задач, связанных с проблемой 2000 года. В то же время, в "Сталкер" заложено множество перспективных возможностей. Центр Компетенции ЛАНИТ использует также базу знаний EDS Vendor2000 для

получения информации об аппаратуре и системном программном обеспечении и Compuware QACenter для автоматизации тестирования модифицированного комплекса. Таким образом, Центр Компетенции ЛАНИТ предоставляет полный спектр услуг по решению проблемы 2000 года в информационных системах.

Обучение методам решения проблемы 2000 года

Одной из существенных мер по преодолению проблемы 2000 года является создание центров для массового обучения специалистов в этой области.

Для технических специалистов нужны глубокие знания предметной области; такой специалист при необходимости должен уметь решить проблему на каждом из уровней, начиная от аппаратных средств и заканчивая внесением изменений в тексты прикладных программ. Наличие большого количество специалистов такой квалификации в большой мере гарантирует предприятию или учреждению успешное решение проблемы 2000 года. В Центре Компетенции по проблеме 2000 года ЛАНИТ проводится специальный курс для подготовки специалистов по проблеме 2000 года. В его рамках рассматривается сущность проблемы 2000 года применительно к техническим и программным средствам, изучается методология решения проблемы 2000 года, освещаются основные способы решения задач, связанных с техническими и программными средствами. Кроме того, прошедшие обучение специалисты получат знания, необходимые для составления планов действий по решению проблемы 2000 года на всех этапах. В курсе также широко описаны программные и технические средства, используемые на каждом этапе решения проблемы 2000 года. Курс рассчитан на специалистов отделов автоматизации/информатизации, отвечающих за работоспособность информационной системы.

Однако для успешного решения проблемы в масштабе всей организации необходимо грамотное планирование всего комплекса работ. Еще один курс, организованный Центром Компетенции ЛАНИТ, рассчитан на руководителей организаций и начальников отделов автоматизации/информатизации. Его слушатели изучают сущность проблемы 2000 года, российские нормативные документы, относящиеся к проблеме 2000 года, методологию решения проблемы 2000 года, учатся привлекать внутренние и внешние ресурсы для решения проблемы и составляют обоснованный план действий по решению проблемы 2000 года применительно к своей организации. От слушателей данного курса никакой специальной подготовки не требуется. Этот курс также полезен для специалистов консалтинговых фирм.

Наконец, в Центре Компетенции ЛАНИТ проводятся специальные курсы по инструментальным средствам реинжиниринга. Его слушатели получают навыки использования современных систем реинжиниринга для анализа информационной системы и оценки сложности ее переноса в новое окружение, учатся использовать современные системы автоматизации реинжиниринга для модернизации информационной системы и решать проблему 2000 года с помощью систем реинжиниринга.

Все три курса сертифицированы и рекомендованы Государственным Комитетом по информатизации и связи. Курсы читаются на базе крупнейшего в России центра авторизованного обучения "Сетевая Академия ЛАНИТ" и проводятся высококвалифицированными специалистами. Подробная информация об этих курсах может быть найдена на Web-странице "Сетевой Академии" – www.academy.ru

Заключение

На базе холдинга ЛАНИТ в целях выполнения Распоряжения Правительства Российской Федерации №671-р создан Центр Компетенции №1 по проблеме 2000 года. Параллельно с выполнением конкретных заказов по решению проблемы 2000 года Центр Компетенции ведет постоянное совершенствование своих методик и инструментальных средств, накопление информации о типовых ошибках, расширение списка имен "подозрительных" переменных, расширение списка анализируемых языков программирования и т.д. Центр Компетенции ведет переговоры с ведущими фирмами-изготовителями системного ПО и аппаратных средств с целью получения скидок при массовых закупках их продукции и организации совместных работ по решению проблемы 2000 года.

Недавно Центром Компетенции ЛАНИТ был выпущен пакет программного и методического обеспечения по решению "Проблемы 2000 года" ***Lan2000***. Пакет Lan2000 содержит средства автоматизации решения проблемы 2000 года как в аппаратных, так и в программных платформах и предназначен для малых и средних предприятий – в Lan2000 входят тесты компьютерных часов LanRTC и LanBIOS, система "Инвентаризация 2000", автоматизирующая процесс сбора информации об используемых на предприятии системах, и система "Сапер 2000", автоматизирующая процесс решения проблемы 2000 года в прикладных программах. Кроме того, в Lan2000 включены методические материалы Центра Компетенции ЛАНИТ по решению проблемы 2000 года.

За 6 лет работы нами накоплен большой опыт по решению проблемы 2000 года. Холдинг ЛАНИТ обладает полным набором инструментальных средств для решения проблемы 2000 года. Кроме того, являясь одним из крупнейших системных интеграторов России, холдинг ЛАНИТ имеет уникальный потенциал для решения проблемы 2000 года в корпоративных информационных системах.

Наши координаты

На Web-страницах 2000.eds.lanit.ru и www.tepkom.ru/y2k можно найти подробную информацию о деятельности Центра Компетенции ЛАНИТ. Телефон в Москве: (095) 967-6650, телефон в Санкт-Петербурге: (812) 428-4187.

Авторам статьи можно написать по адресу: y2k@tepkom.ru

Литература

[1] **А.Н.Терехов, А.А.Терехов** *Перенос приложений и проблема 2000 года* // Компьютер-Пресс. – 1998. – №8, стр.92-96
(см. http://www1.tepkom.ru/users/ddt/Articles/Article.html)
[2] **А.Колесов** *Тень на плетень, или Задача для начинающих программистов* // "Известия", 28 августа 1998 года, стр.4
[3] **Jones C.** *Global economic impacts of the year-2000 problem* // Year-2000 Problem: Strategies and Solutions from the Fortune-100, L.Kappelman, Ed. – Boston, 1997, International Thomson Press, P.12-29
[4] **Kappelman L.A., Fent D., Keeling K. B. Prybutok V.** *Calculating the Cost of Year-2000 Compliance* // Communications of the ACM. – 1998. – Vol. 41, No.2. – P.30-39.
[5] **Lefkon D.** *Seven Work Plans for Year-2000 Upgrade Projects* // Communications of the ACM. – 1997. – Vol.40, No.5. – P.111-113
[6] **R.A.Wagner** *Solving the Date Crisis* // Communications of the ACM. – 1997. – Vol.40, No.5. – P.115-117
[7] **Г.Гинкул, С.Соловьев, А.Сотников** *Проблема 2000 года: методы корректировки дат* // Вопросы статистики. – 1998. – №9, стр.68-73
[8] **М.П.Березка** *Опыт решения проблемы 2000 года для системы резервирования мест и продажи железнодорожных билетов "Экспресс-2"* // Семинар "Проблема 2000. Подход на уровне правительства", Москва, 26-27 ноября 1998 г.
[9] **Чарльз Бруно** *"Ошибка тысячелетия" может стать и сетевой проблемой* // Сети. – 1998. – №1 (см. http://www.osp.ru/nets/1998/01/2000.htm)
[10] **Исмо Саволайнен, Жанет Борзо** *Финские власти проверяют готовность российских АЭС к 2000 году* // ComputerWorld Россия. – 1998. – №33.
(см.http://www.osp.ru/cw/1998/33/02.htm)
[11] **Боб Бруин** *Ядерные державы объединяются против 2000 года* // Computerworld Россия. – 1998. – №24.
(см. http://www.osp.ru/cw/1998/24/35.htm)