Кириченко А.А.

Технологии экстремального программирования

2014

Кириченко А.А.

«Технологии экстремального программирования», 2014. Сетевое электронное издание монографии. 161 страница, формат PDF.

ISBN 978-5-9904911-3-7

Рецензент: к.т.н., доцент Гудыно Л.П.

Экстремальное программирование – это программирование в условиях, для которых характерны крайне ограниченные ресурсы (время, кадры, финансирование и др.), а требования к функциям, возможностям, производительности и другим техническим характеристикам значительно превышают значения, которые они могли бы иметь в нормальных условиях.

Практика показывает, что для успешного выхода из экстремальной ситуации надо увеличить скорость и качество разработки программ, для чего необходимо повышение квалификации членов группы программистов, т.е. уровня их знаний и умений. В связи с этим в монографии рассматриваются факторы, способствующие повышению эффективности работы программистов, инструментарий, облегчающий создание программ и приёмы рационального построения процесса программирования с использованием компонентов, использование системы команд операционной системы из алгоритмического языка высокого уровня, комплексирование программных средств с помощью командного файла операционной системы, технология экстремального программирования Кента Бека. Заканчивается рассмотрение способов ускорения процесса программирования перспективами развития технологий экстремального программирования.

Монография может быть полезна студентам, бакалаврам, магистрам, аспирантам специализирующимся на использовании компьютеров при решении экономических задач.

Кириченко А.А. профессор департамента программной инженерии факультета компьютерных наук Федерального государственного автономного образовательного учреждения высшего профессионального образования “Национальный исследовательский университет "Высшая школа экономики" при правительстве РФ”.

ISBN 978-5-9904911-3-7

Содержание.

Введение

Определение понятия «Экстремальное программирование».

Технологии экстремального программирования – это дисциплина, относящаяся к разделу «Технология программирования», но имеет интригующее уточнение: не просто программирования, а экстремального программирования! Поэтому первый вопрос, на который надо ответить, что такое экстремальное программирование?

Будем считать, что экстремальное программирование – это программирование в экстремальных условиях.

Какие условия относятся к экстремальным? Эдвард Йордон в книге «Путь камикадзе» экстремальное программирование приравнивает к работе над “безнадёжным” проектом, для которого характерны:

- неспособность высшего руководства воспринимать правдивую информацию о проекте,
- отсутствие у проектировщиков пространства для манёвра в отношении функциональности, затрат или времени:
    - план проекта сжат более, чем наполовину по сравнению с нормальным расчётным планом;
    - число разработчиков уменьшено более, чем наполовину по сравнению с действительно необходимым для выполнения проекта данного размера и масштаба;
    - бюджет и связанные с ним ресурсы урезаны наполовину;
    - требования к функциям, возможностям, производительности и другим техническим характеристикам вдвое превышают значения, которые они могли бы иметь в нормальных условиях.

Из всего сказанного надо отметить:

- неспособность высшего руководства воспринимать правдивую информацию о проекте,
- отсутствие у проектировщиков пространства для манёвра в отношении функциональности,
- недостаток времени,
- крайне ограниченные ресурсы.

Софтверные проекты характеризуются большим разнообразием, и полное и строгое соответствие перечисленным характеристикам не всегда обязательно. По-другому охарактеризовать безнадёжный проект можно тем, что вероятность его провала превышает 50%.

Служба Microsoft Consulting Services провела анализ результатов выполнения большого количества своих программных проектов.

Оказалось, что:

- только 24% проектов можно признать в той или иной степени успешными,
- 26% не были завершены,
- а остальные 50% столкнулись с большими проблемами, например, бюджет был превышен вдвое или затрачено в 1,5 раза больше времени.

Основными причинами неудач были признаны следующие:

- постоянное изменение требований;
- нечеткие или неполные спецификации;
- низкое качество кода;
- слишком широкая постановка задачи;
- ошибка в подборе кадров;
- плохая организация работы;
- нечетко сформулированные цели.

Возможные варианты выхода из экстремальной ситуации.

Практически каждый программист рано или поздно попадает в экстремальные условия. И когда он, наконец, из них выходит с победой, ему очень хочется поделиться со всеми своим опытом. Наверное этим можно объяснить появление более сотни методик экстремального программирования.

Во многом эти методики пересекаются, дублируют друг друга. Поэтому среди них можно выделить относительно самостоятельные три – четыре, которые принято относить к разряду гибких технологий:

- Экстремальное программирование (XP);
- Унифицированный процесс разработки;
- Гибкое моделирование.

Нельзя считать, что решением этих проблем не занимались ранее. Известны такие методы организации программирования, как

- нисходящее проектирование,
- структурное программирование
- визуальное (или событийное) программирование
- компонентно-ориентированное программирование
- объектно-ориентированное программирование.

При ограниченных временных ресурсах выход можно найти за счёт ускорения процесса программирования, с которым обычно связаны:

- организация проектирования,
- работа группой программистов,
- автоматизация программирования,
- комплексирование программных средств, (то есть сборка в одном проекте всех имеющихся в наличии средств, пригодных для решения поставленной задачи),
- упрощение решаемой задачи,
- менее строгий подход к выбору метода решения задачи (т.е. реализации принципа: выдать результат любым путём, не искать оптимальных алгоритмов, не исследовать варианты, …).

На организацию проектирования, эффективность работы группы программистов оказывают влияние

- квалификация сотрудников,
- психологическая совместимость группы,
- организация общения партнёров (особенно при использовании удалённых кадров).

Все эти параметры усиливаются спецификой, накладываемой рыночными условиями.

Особенности программирования в рыночных условиях.

Терехов А. Н. в книге «Введение в технологию программирования» (http://www.intuit.ru/department/se/introprogteach/) обращает внимание на особенности программирования в рыночных условиях:

«В 1976 году мне пришлось довольно долго беседовать с руководителем разработки трансляторов с языка ПЛ1 для IBM/360 по фамилии Маркс. Родился он в тот же год, что и я, закончил университет тогда же, когда и я, занимался очень похожей работой (в то время я был одним из руководителей разработки транслятора с языка Алгол 68), в общем, нам было интересно побеседовать. Я его спросил: "Почему PL1/F имеет 51 просмотр, когда с Алголом 68 мы справляемся за 6?". Он отвечает: "А что тут понимать, у меня в подчинении был 51 программист. К тому времени, когда мы начали реализовывать оптимизирующий транслятор с ПЛ1, у меня было 100 человек, поэтому PL1/ opt имеет 100 просмотров".

Я и тогда понимал, что это глупо. Между каждой парой просмотров нужно организовывать файл на промежуточном языке; один просмотр – пишет, другой – читает, добавьте разные упаковки и распаковки, другие служебные действия – и вы поймете, почему трансляторы с ПЛ1 работали так медленно.

Но не все так просто в этом мире.

Мы делали транслятор чуть ли не 10 лет, а они "слепили" свой транслятор за два года.

Тот же Маркс позавидовал мне, что у нас есть такая замечательная возможность заниматься любимой наукой, не торопясь, исследовать разные варианты, придумывать новые методы, публиковать монографии и т.д.

С понятием рынка ПО в те годы мы были совершенно не знакомы, **да и сейчас, через 30 лет, далеко не все наши программисты понимают законы рынка:**

как важно первым выдать на рынок пусть не самый лучший, но работающий продукт, соответствующий определенным потребностям рынка».

Таким образом, в рыночных условиях программирование – это не искусство, не наука, а производственный процесс, в котором упростить выход из экстремальной ситуации позволяют четыре фактора:

- Методы управления работой группы программистов;
- Квалификация членов группы, т.е. уровень их знаний и умений;
- Применяемые технологии программирования;
- Средства создания, отладки и тестирования программ.

Практика показывает, что в конечном итоге для успешного выхода из экстремальной ситуации надо увеличить скорость и качество разработки программ.

Для этого надо

- уметь работать в команде,
- программировать на различных алгоритмических языках (или хотя бы – уметь их использовать в своих целях),
- владеть современным инструментарием для создания программ,
- знать и уметь применять на практике различные системы и технологии программирования,
- уметь строить системы опережающего тестирования и владеть средствами автоматизации этой работы,
- иметь навыки отладки программ.

Умение работать в команде предусматривает наличие знаний:

- Как подбираются кадры для командной работы, каким образом можно сформировать эффективно работающую команду;
- Как управлять коллективом, как коллектив развивается, методы управления коллективом;
- Какие типы команд существуют;
- Как организуется общение в команде, в том числе – в команде, члены которой территориально разделены, например, общение через Интернет,
  - с помощью отдельного сайта совместного проектирования,
  - с помощью системы взаимосвязанных сайтов,
  - используя Интернет-телефонию;
- Как организуется обмен информацией в команде. Роль документации в программировании. Технология представления документации в виде
  - линейного документа Word или Power Point,
  - интерактивного документа,
  - документа типа Web-сайта,
  - html-help – документа,
  - XML – документа;
- Инструментарий для создания различных видов документов.

Владение инструментарием для создания программ предусматривает

- освоение дополнительных разделов программирования на современных алгоритмических языках, таких как C#,
- активное использование компонентного программирования и библиотек программ,
- свободное использование динамических библиотек и ресурсов, что создаёт предпосылки для
  - комплексирования программных средств, т.е. широкого использования имеющихся программных средств, выполненных на других языках, в других проектах, и тем самым – сокращения времени выполнения проекта.
- умение тестировать и отлаживать программы,
- использовать средства автоматизации тестирования и отладки, которые так же способствуют сокращению времени разработки программных проектов.

Постановка задачи

Исследование возможных вариантов выхода из экстремальной ситуации показывает, что для выхода из экстремальной ситуации программисту необходимо освоить методы и приёмы ускорения программирования за счёт:

- более глубокого знания языка, на котором ведётся программирование,
- за счёт расширения возможностей концепции используемой системы программирования, разработки новых компонент, и др.
- комплексирования программных средств,
- профессионального владения поиском и добычей информации
- совершенствования технологии программирования, организации работы группового программирования.

В монографии собраны способы, средства и методы, способствующие ускорению программирования. Они распределены по следующим разделам:

1. Эффективность работы программистов.
   - Поиск и добыча информации.
   - Работа с помощью сайта совместного проектирования
   - Организация совместной работы с помощью Интернет-телефонии
   - Использование e-mail для поиска и получения информации с помощью почтовых роботов
   - Различные способы изготовления документации.
   - HTML-help файлы, как средства создания архивов документов.
2. Инструментарий, облегчающий создание программ
   - Сборка C# - проекта, использование системы команд компилятора csc (C#-compiler)
   - Основные принципы использования ресурсов в создаваемых сборках
   - Приёмы работы с ресурсами.
3. Программирование с использованием компонентов
   - Библиотека MSDN (Microsoft Developer Network), содержащая информацию о программировании с использованием компонентов
   - Примеры кода для программирования компонентов
4. Использование системы команд операционной системы (ОС) из C#-программы
   - Классы .NET для работы с процессами
   - Запуск команд ОС из C#-программы
5. Комплексирование программных средств с помощью командного файла ОС (command, CMD-файла).
6. Технология XP (eXtream Programming) Кента Бека.

- Историческая справка
- Структура идеального XP-процесса
- Стратегическое и тактическое планирование
- Игра в планирование
- Планирование версий

7. Перспективы развития технологий экстремального программирования.
   - Автоматизация программирования.
   - Моделирование мышления.
   - Нетрадиционные подходы.

В приложениях содержатся материалы для подготовки некоторых практических занятий:

Приложение 1. TeamSpeak.

Приложение 2. Упрощённая методика создания многостраничного сайта.

Приложение 3. Обзор средств создания HELP - файлов

Приложение 4. Техника работы с мастером Windows Application.

Эффективность работы программистов.

Поиск и добыча информации.

Поиск информации в Интернет обычно производится с помощью различных поисковых машин. Чаще всего используются словарные поисковые машины, представителем которых является Google.

Технология поиска информации описана в книге "Google Hacks", написанной Tara Calishain и Rael Dornfest (Реквизиты: Publisher : O'Reilly Pub Date : February 2003 ISBN : 0-596-00447-8 Pages : 352).

Частичный перевод книги можно найти по адресу:
http://elijah.poetry.com.ua/texts/google.html

Базовая поисковая техника:

Для поиска требуемой информации составляется поисковое предписание, содержащее характеризующие эту информацию ключевые слова.

Дополнительные правила составления поискового запроса:

- Google не воспринимает более 10 слов для поиска, включая специальный синтаксис.
- Знак (+) в поисковом предписании используется для поиска наиболее общих слов (пробелы между знаком и помечаемым этим знаком словом не допускаются). Часто используемые слова "I", "a", "the", "of" и т.п. игнорируются поисковиком, но можно заставить его их искать если поставить перед ними "+". Например: "Война +и мир"
- Знак (-) в поисковом предписании используется для исключения термина из поиска (пробелы между знаком и помечаемым этим знаком словом не допускаются)
- Для поиска фразы надо поместить её в кавычки «»
- Знак «*» означает любое слово. Google не поддерживает поиска по корням слов (stemming), то есть возможности использования звёздочки (или другого знака маски) вместо букв в искомом слове. Например, moon* в поисковике, поддерживающем маски, найдёт "moonlight," "moonshot," "moonshadow," и т.д. Google же использует звёздочку как заменитель целого слова. Поиск по фразе "three * mice" в Google даст в результате "three blind mice," "three blue mice," "three red mice," и т.д.

- Обойти лимит в 10 слов можно используя звёздочки. Каждая звёздочка заменяет одно слово. Как оказалось, Google просто не считает количество звёздочек в запросе.
  "do as * say not as * do" quote origin English usage - замена "do as I say not as I do" quote origin English usage

Так как Google является полнотекстовым поисковиком, он индексирует всё содержимое страниц. Дополнительные команды, называемые спец. синтаксисом (операторы для продвинутого поиска) позволяют пользователям Google искать конкретные части web страниц или тип информации. Это позволяет сузить число результатов поиска.

Такие операторы имеют следующий синтаксис: «operator:search_term» (в этом выражении также не должно быть никаких пробелов).

Примеры этих операторов:

- **site:** инструктирует Google ограничить поиск конкретным web-сайтом (доменом); название сайта (домена) указывается сразу после двоеточия и без пробела. "site:" сужает поиск до одного сайта или домена верхнего уровня. Например:

site:loc.gov
site:thomas.loc.gov
site:edu
site:nc.us

- **filetype:** инструкция произвести поиск только в пределах текста указываемого типа файлов. Тип файла указывается после двоеточия (точку перед расширением файла указывать не нужно). "filetype:" ищет среди расширений файлов, а точнее - в файлах с определённым расширением. Например:

homeschooling filetype:pdf
"leading economic indicators" filetype:ppt

- **link:** производить поиск внутри гиперссылок содержащих поисковый запрос. "link:" возвращает список страниц, имеющих ссылку на заданную. Например, введите "link:www.google.com" и увидите список сайтов имеющих ссылку на Google. Не обязательно вводить "http://"; Google проигнорирует эту часть текста даже если её ввести. "link:" отлично работает как с "глубокими" адресами, вроде "http://www.raelity.org/apps/blosxom/", так и с верхнеуровневыми URL, такими как "raelity.org".
- **cash:** оператор демонстрирует версию страницы, которая существовала, когда она индексировалась Google. URL страницы указывается сразу после двоеточия. "cache:" ищет копию страницы проиндексированной Google даже если страница уже недоступна по оригинальному URL или её содержимое полностью изменилось. Например:

cache:www.yahoo.com

- **intitle:** производить поиск внутри названия документа. "intitle:" ограничивает поиск до заглавий страниц (titles). Вариации, "allintitle:" ищет страницы в заглавии которых находятся все слова поиска. Например:

intitle:"george bush"
allintitle:"money supply" economics

- **inurl:** искать внутри URL документа. "inurl:" ограничивает поиск до адресов (URL) страниц. Команда хороша для поиска страниц помощи и поиска, так как они имеют довольно стабильную структуру. "allinurl:" вариация, которая ищет все введённые слова в URL. Например:

inurl:help
allinurl:search help

- **intext:** ведёт поиск только по тексту страниц (т.е. игнорирует текст ссылок, URL, и заглавий). Есть вариация "allintext:", но она плохо ладит с другими командами. Например:

intext:"yahoo.com"
intext:html

- **inanchor:** ищет текст в якорях ссылок на страницах (anchors). Якори ссылок – это текст описания ссылки. Например, во фрагменте кода HTML [a href="http://www.oreilly.com" O'Reilly and Associates] якорем ссылки является "O'Reilly and Associates." Пример:

inanchor:"tom peters"

- related: находит страницы, похожие на запрашиваемую. Например, поиск "related:google.com" даст множество поисковиков, включая HotBot, Yahoo!, and Northern Light. Аналогично:

related:www.yahoo.com
related:www.cnn.com

- info: предоставляет ссылки на более подробную информацию о запрошенном URL. Информация включает ссылки на кэш URL, список страниц имеющих ссылки на данную, страницы, связанные с данной, страницы, содержащие данный URL. Например:

info:www.oreilly.com
info:www.nytimes.com/technology

Такие операторы можно использовать как по отдельности, так и в различных сочетаниях, в том числе – с различными ключевыми словами. Но некоторые из них отлично работают в сочетании друг с другом, некоторые друг другу мешают, а некоторые просто друг с другом не работают.

Индивидуальные, не сочетающиеся с другими, команды: rphonebook:, bphonebook:, phonebook:, link:. Остальные спец. команды можно смешивать как угодно.

Карта сайта

Чтобы выявить каждую страницу на сайте, Google сканирует его, используя оператор "site:" и дополнительные ключевые слова, которые должны содержаться на **каждой** странице сайта.

Например, составим запрос такого вида: **site:http://www.microsoft.com microsoft**. Этот запрос выполняет поиск по слову «microsoft» в пределах сайта http://www.microsoft.com.

Как много страниц на сервере Microsoft содержат слово «Microsoft»? Для выяснения этого вопроса надо иметь в виду, что Google исследует не только содержание страниц, но также их название и URL. Слово «Microsoft» стоит в URL каждой страницы http://www.microsoft.com.

Таким образом - единственным запросом можно инициировать обработку каждой страницы на сайте Microsoft, проиндексированной Google.

Нахождение листинга директории

Листинг директории представляет собой список файлов и директорий удаленного сервера в окне браузера. Такие листинги открывают широкие возможности для углубленного сбора информации. Как правило: такие страницы директорий имеют в Title и теле страницы выражение «Index Of». Отсюда очевидно и строение запроса для поиска таких листинов - это «intitle:index.of». В результате такого запроса будут найдены страницы со словом «index of» в разделе Title документа.

К сожалению – этот запрос вернет слишком большое число страниц не по теме, к примеру, страницы вида:
• Index of Native American Resources on the Internet
• LibDex—Worldwide index of library catalogues
• Iowa State Entomology Index of Internet Resources

Исходя из названий найденных документов, очевидно, что эти страницы не соответствуют заданному запросу и вряд ли окажутся искомыми списками директорий.

Следующие запросы обеспечат более точные результаты:
intitle:index.of "parent directory"
intitle:index.of name size

Такие запросы более точно выдадут то, что нам нужно, поскольку ориентированы не только на фразу «index of» в Title страницы, но и на ключевые слова, всегда имеющиеся в листингах директорий: «parent directory», «name», «size».

Определение версии WEB-сервера

Точная версия программного обеспечения web сервера – это один из элементов, необходимых администратору для точной настройки Web-сайта. Если непосредственно соединиться с сервером, то HTTP (web) заголовки (headers) этого сервера предоставят нужную информацию. Однако можно получить эту информацию **из кэша Google безо всякого соединения с сервером**. Такой метод основан на использовании списка директорий.

Список файлов директории включает имя серверного софта и его версию.

Выглядит такой запрос просто: **«intitle:index.of server.at»**

Он основан на содержании фразы **«index of»** в разделе **title** страницы директории и фразы **«server.at»**, содержащейся в конце любого листинга директории. К примеру, так выглядит запрос, определяющий версию сервера aol.com:

«intitle:index.of server.atsite:aol.com».

Использование Google в качестве сканера CGI директорий

Для выполнения подобной задачи, CGI сканер изначально знает - какие именно директории нужно искать на сервере. Как правило - это директории, в которых располагаются файлы данных и имеют вид, подобный представленным ниже:
/cgi-bin/cgiemail/uargg.txt
/random_banner/index.cgi
/random_banner/index.cgi
/cgi-bin/mailview.cgi
/cgi-bin/maillist.cgi
/cgi-bin/userreg.cgi
/iissamples/ISSamples/SQLQHit.asp
/iissamples/ISSamples/SQLQHit.asp
/SiteServer/admin/findvserver.asp
/scripts/cphost.dll
/cgi-bin/finger.cgi

Зная синтаксис требуемых директорий, а также владея техникой поиска, изложенной выше, можно использовать Google как CGI сканер.

- Например, поиск в Google следующего вида: **allinurl:/random_banner/index.cgi** вернет документы с адресами страниц конкретных программ генерации рекламных баннеров.

Использование Google как внутренней поисковой системы Web-сайта.

При поиске информации на серверах можно использовать не только их "родные" формы поиска, но и Google. Например? Поисковый запрос:
""george bush" site:nytimes.com" - поиск статей про Дж.Буша на сайте Нью Йорк Таймс.

Сайт совместного проектирования

Г.Майерс в книге «Надёжность программного обеспечения» отмечает: «Опрашивая группу руководителей о признаках хорошего программиста, можно услышать о таких качествах, как способности к анализу, распознаванию образов, алгоритмическому мышлению, концентрации и дедуктивным рассуждениям.

При опросе группы более опытных руководителей могут появиться такие факторы, как профессионализм, подготовка в области оснований информатики, способность работать в напряжённых условиях, способность работать с другими людьми, умение приспосабливаться к изменениям, аккуратность, стремление к совершенствованию, и даже чувство юмора.

Всё это, конечно, требуется в работе программиста, но, возможно, самый существенный фактор это *способность к общению*.

Программисты тратят около двух третей своего времени на общение (слушают, читают, разговаривают и пишут), а … большинство ошибок в программном обеспечении связано с проблемами взаимного общения между людьми.

Этим объясняется интересный феномен: выпускники с дипломами по литературе или музыке (которые в первую очередь требуют мастерства в общении) часто оказываются выдающимися программистами.

Главным вопросом при найме на работу и подготовке программ обучения должно стать умение общаться, т.е. умение программиста выразить свои собственные мысли и понимать идеи других».

В современных условиях совместно работающие программисты часто бывают территориально разделены. Но при использовании достаточно простых и доступных средств, проблема коммуникации решается достаточно просто.

К наиболее распространённым и самым доступным способам коммуникации относится электронная почта. Но она является способом преимущественно общения двух лиц. И этот канал относится к симплексному типу: один передаёт, другой слушает (или читает).

Менее распространённым, но таким же доступным являются телеконференции. Правда, и у них есть свои недостатки, затрудняющие их использование одной и той же группой лиц постоянно и непрерывно.

Видимо, самым удобным, доступным и дешёвым является общий Web-сайт. Его можно создать на одном из бесплатных хостингов, и не регистрировать в поисковых системах. Спрятать такой сайт легко можно, создав несколько входов на него. Один из этих входов – стандартный, выполненный в виде файла index.html. С этой страницы вход внутрь сайта не делается, она является заглушкой. Настоящий вход делается в виде файла, например, start.html.

Для изготовления такого сайта нет необходимости в использовании специальных средств – достаточно Word. В Приложении 2 приведена методика создания многостраничного сайта. Общий Web-сайт профессионалы называют Web для совместного проектирования. Этот Web предназначен для организации совместной работы специалистов, находящихся на значительном расстоянии друг от друга. Для совместной работы необходимо иметь общий, согласованный со всеми план действий. Каждый участник должен знать, с какой целью ведется данная работа, какую часть общей работы кто выполняет, в какие сроки должна быть выполнена та или иная часть работы, какие промежуточные результаты получены другими участниками работы. На таком Web

помещается оперативная информация, связанная с общей работой. Если при выполнении проекта необходимо совместное использование имеющихся в наличии ресурсов, на Web размещаются средства, позволяющие согласовывать, кто, когда, в каком объеме будет использовать эти ресурсы. Это «место кучкования» участников совместного проекта.

Сайт может использоваться для работы с партнерами фирмы, поставщиками комплектующих, для выполнения научно-исследовательских работ, для проведения социологических исследований. Такой сайт часто используют программисты – для совместной разработки, тестирования программных средств.

Для совместной реализации проектов удаленными исполнителями используется специальная организация веб-сайта и может потребоваться специальное программное обеспечение типа персональных информационных менеджеров (PIM – Personal Information Manager). Фирма Microsoft предлагает для проектирования таких сайтов использовать пакет Front Page.

В состав Microsoft FrontPage входит шаблон для создания сайтов, благодаря которому за несколько минут можно создать сайт Web-проектов. Чтобы получить доступ к шаблону необходимо выбрать пункт меню File->New->Web, после чего в появившемся окне выбрать шаблон Project Web.

Навигационная структура такого сайта представлена на рисунке:

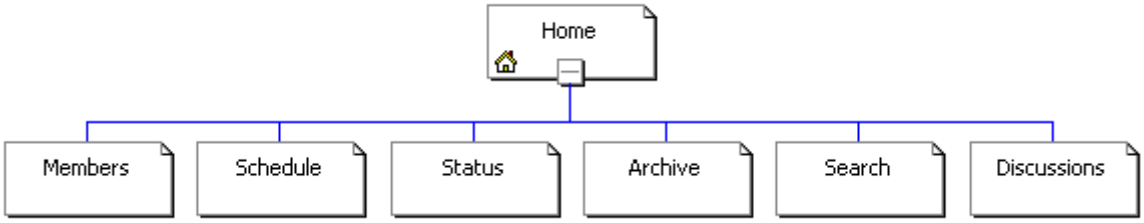


Рис. 1. Навигационная структура сайта "Web проекта".

Файловая структура представлена на рисунке:

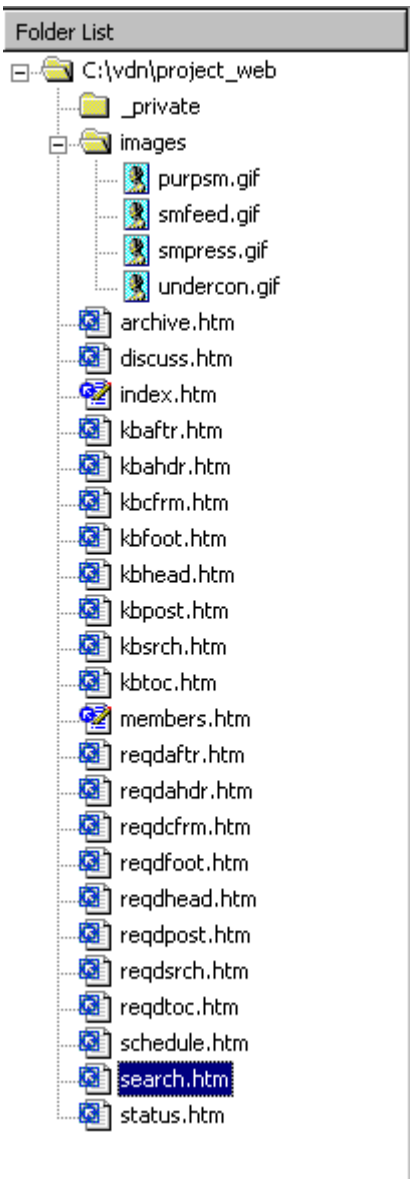


Рис. 2. Файловая структура сайта "Web проекта".

Типовая пользовательская карта сайта, предназначенного для ведения Web-проектов приведена на рисунке:

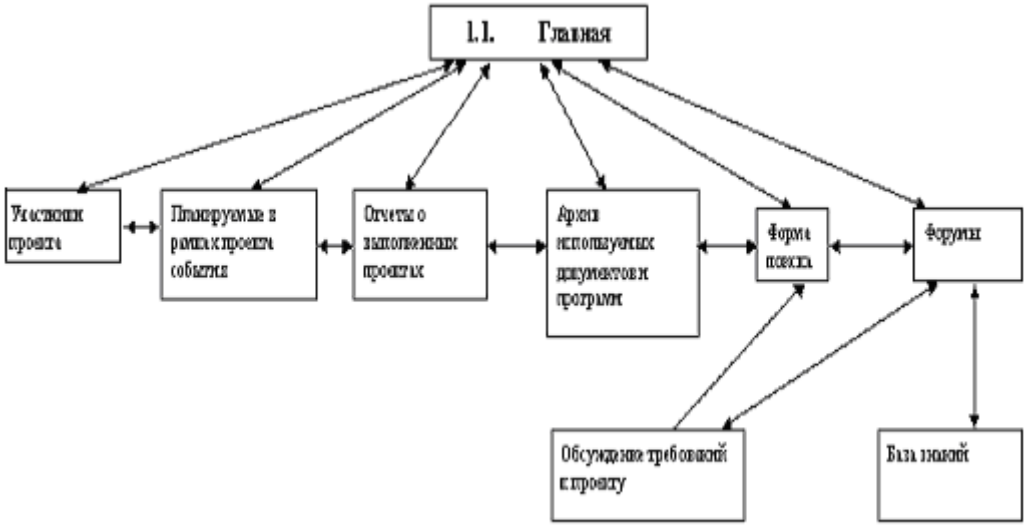


Рис. 3. Типовая пользовательская карта сайта "Web проекта".

В разделе **“Участники проекта (Members)”** обычно приводится контактная информация о всех лицах, которые задействованы в проекте (профессия, местоположение офиса, e-mail, телефон, URL сайта).

В разделе **“Планировщик (Shedule)”** приводится список всех важных в рамках проекта событий в хронологическом порядке.

В разделе **“Состояние (Status)”** приводятся месячные, квартальные и годовые отчеты о ходе выполнения различных частей проектов. В начале каждого месяца информация обновляется.

В разделе “**Архив используемых документов и программ (Archive)**” участники проекта имеют доступ к проектной документации, компонентам системы, а также к используемому ПО и вспомогательным утилитам.

В разделе **“Форум (Discussions)”** участники проекта обмениваются информацией о ходе выполнения проекта, а также формируют корпоративную базу знаний.

При внимательном рассмотрении структуры такого сайта можно оптимизировать его для совместной разработки конкретных программ.

Решение той же задачи можно выполнить за счёт создания куста сайтов, обеспечивающего работу бригады программистов при значительном удалении членов бригады друг от друга. В кусте каждый член бригады имеет доступ к своему сайту, и на каждом сайте есть гиперссылки, позволяющие выйти на сайты всех остальных членов бригады. Структуру куста нужно создавать с учётом того, какой информацией и в каком направлении должны обмениваться члены бригады.

Интернет-телефония

С развитием глобальной сети Интернет, разработчики программ для живого общения в сети стали задумываться: «как облегчить потребителю общение в удовольствие?». Одними из первых, на этот вопрос ответили разработчики такой линии программ общения, как Niels Werensteijn (Разработчик клиентской версии программы TeamSpeak) и Ralf Ludwig (Разработчик серверной версии программы TeamSpeak). Они

представили на общее обозрение и пользование свою программу – TeamSpeak. Именно в TeamSpeak был реализован принцип использования голосового общения с помощью наушников и микрофона, для чего необходимо установить программу и зайти на канал в Интернете, либо создать собственный канал.

Существует два типа программы TeamSpeak – серверная и клиентская. Серверный тип предназначен для пользователей, которые решились создать свой собственный канал, и приглашать на него посетителей. Клиентский тип представляет собой программу, которая присоединяется к уже имеющемуся каналу (может быть расположен как в Интернете, так и на Сервере).

К явным плюсам программы TeamSpeak можно отнести:

- Поддержка мульти сессии (можно общаться с группой человек одновременно)
- При общении не нужен телефон или другие устройства связи (чаты)
- Программа не требовательна к операционной системе (работает под старыми операционными системами)
- Хорошее качество передаваемого/получаемого звука (использует разные кодеки звука)
- Программа не прихотлива к трафику (весьма полезно для тех, кто использует модемные соединения с Интернет)
- При отсутствии микрофона, можно слушать звук в колонках (только слушать, отвечать не сможете)
- Удобный и не сложный интерфейс (нужны минимальные знания английского языка)
- Небольшой размер дистрибутива программы (Клиентский тип – 5.59 Mb, Серверный тип – 1.48 Mb)
- Совместимость с другими запущенными программами (игры через Интернет, фильмы, выполнение общих работ)
- Бесплатное распространение (Freeware)

При установке сервера в Интернет на хост-компьютере, он будет иметь постоянный IP-адрес, по которому с ним могут связываться клиенты. Этот вариант требует материальных затрат, так как размещать серверную часть TeamSpeak на чужом хосте можно только с разрешения его владельца.

Машины-клиенты постоянного IP-адреса не имеют. Временный адрес им выделяется хост-компьютером провайдера при подключении к сети – и только на время сеанса. Для следующего сеанса клиенту может быть выделен другой адрес.

Серверную часть пакета можно установить на машине-клиенте. Если известен временный IP-адрес этого клиента, то во время сеанса (т.е. когда машина с серверной частью пакета TeamSpeak подключена к Интернет), с ним можно связаться и работать.

Таким образом, при размещении сервера TeamSpeak на одной из машин-клиентов необходимо:

- определить временный IP-адрес машины, на которой размещён сервер TeamSpeak;
- сообщить этот IP-адрес всем участникам общения.

После этого все участники общения с помощью установленных у них клиентов TeamSpeak связываются с сервером и организуют полноценное речевое общение, облегчающее групповое выполнение каких-либо работ или задач.

Использование e-mail для поиска и получения информации с помощью почтовых роботов

Почтовым роботом называется программа, которая

- получает электронную почту, приходящую в ее адрес,
- определяет, на какую информацию был прислан запрос

– и выполняет необходимые действия по выполнению пришедшего запроса.

Полученные результаты отправляются на e-mail запросившего информацию клиента.

Получить инструкцию по работе с почтовым роботом Archie, действующим на хост-компьютере archie.doc.ik.ak.uk можно следующим образом:

- создаем новое письмо:
- в поле “TO (кому)” указываем адрес почтового робота: archie@archie.doc.ik.ak.uk.
- поле “Subjekt (Тема)” оставляем пустым;
- в тексте письма (Message) пишем всего одно слово: help
- отправляем созданное письмо и ждем ответа.

Например,

To: archie@archie.doc.ik.ak.uk
Subject:
Message: help

- Адреса некоторых почтовых роботов:

- **bitftp@pucc.princeton.edu**
- **ftpmail@sunsite.doc.ik.ak.uk**
- **mail-server@rtfm.mit.edu**
- **ftpmail@grasp.insa-lyon.fr**
- **ftpmail@decwrl.dec.com**
- **w3mail@gmd.de**
- **agora@info.lanic.utexas.edu**
- **webmail@curia.ucc.ie**
- **wwwmail@eunet.sk**
- **agora@www.undp.org**
- **web-mail@ebay.com**
- **listserv@info.cern.ch**
- **agora@dna.affrc.go.jp**

- Например, для работы с поисковой системой AltaVista поисковое предписание может выглядеть следующим образом:

get http://www.altavista.com/\web/results?
q=digital+electro\nic+computer&kgs=0&kls=\0&avkw

Почтовый робот своими руками

Почтовые роботы -- это программы, выполняющие определенные действия в ответ на получение электронных писем. В мире Unix они применяются сначала появления глобальных сетей, однако в Windows все еще остаются диковинкой.

Одна из таких диковинок - **Почтовый робот v2.0 beta**

- писалась программа для органов федерального казначейства. Позволяет создавать почтовые сообщения в программах TheBat & Outlook Express (MS Outlook 2000), основываясь на имени файла. Ведётся база адресов. Можно выполнять резервное копирование отправляемых файлов с разбиением по годам, месяцам, дням. Не требует установки.

- Почтовый робот вполне заслуживает того, чтобы стать еще одним инструментом в арсенале программиста, применяемым по ситуации.

Проблема документации.

Довольно широко распространено мнение, что документировать программы совсем не обязательно, достаточно включить в текст комментарии. Это ошибочная точка зрения. Результатом процесса разработки программной системы является документация, описывающая структуру созданной системы (компоненты, подсистемы, программы, процессы) и основные приёмы работы с ней.

Документация нужна на разных этапах создания программ. На некоторых из них она является единственным документом (например, при описании архитектуры системы, исходных текстов программ, руководства по использованию, и т.д.).

Частично функции документации выполняют комментарии, вставляемые в текст программы. Они характеризуют внутреннюю логику программы, определение данных, переменные, их взаимодействие.

Описания результатов, ошибок и методов их устранения в комментариях обычно нет. Программу без документации можно будет использовать только если к ней будет приложен программист, её составлявший. Если программа должна являться коммерческим продуктом, без документации её продать вряд ли получится.

Поэтому совокупность программы и документации к ней называется **программной системой**.

Оптимально документация должна учитывать уровень подготовки пользователя.

Комментарии, включаемые в текст программы, должны составляться так, как будто они являются ответом на вопрос пользователя.

Состав документации определён в нормативных документах, таких, как ЕСПД и ЕСКД.

HTML-help файлы: архивы документов.

Язык HTML позволяет создавать удобные архивы документов, в которых на главной (домашней) странице хранится перечень имеющихся документов. Переход к необходимому документу осуществляется щелчком мышки по требуемому документу. Но оформление архива в виде Web - сайта имеет свои неудобства - создаётся целая папка, содержащая большое количество файлов. Поиск в такой папке файла index.htm иногда представляет сложность. В папку могут попасть случайные файлы, которые увеличат размер архива и создадут дополнительные трудности при работе с ним. Файлы в таком архиве открыты и могут быть подвержены несанкционированной корректировке.

Более удобным средством создания HTML - архивов являются программы для создания HTML - help файлов. Такие программы позволяют создать архив в виде единого файла, который после компиляции не позволяет откорректировать содержание архива. Кроме того, HTML - help файлы могут использоваться по своему основному назначению. Например, в виде такого файла можно оформить отчет, описание созданной системы, и др.

В Windows наиболее популярны два вида Help - файлов: Windows He1p (*.HLP) и HTML-He1p (* .CHM). Более подробный обзор средств создания Help-файлов приведен в приложении 3.

Для создания HTML - архивов используются специальные программные средства, основными среди которых являются **He1p Workshop** и **HTML He1p Workshop –** для файлов справки HLP- и HTML-форматов соответственно.

Наиболее подробная информация о WinHelp'е содержится в документе «Microsoft Windows Help Authoring Guide», который можно скачать в Сети по адресу www.spnet.ru/neuro/help/whag.zip , а описание формата HLP есть в базе данных по форматам – WOTSIT (www.wotsit.org) .

Формат файлов HTML He1p ничего общего не имеет со справочными системами, построенными из обычных HTML - файлов (как, например, это сделано в продуктах GIMP или Photoshop 6).

HTML-He1p - файл открывается при помощи приложения HTML He1p.

Для создания архива документов нужно подготовить базовый текст с рисунками в Word (формат doc или rtf) и импортировать готовое описание в формат htm (можно разбить сохраняемый в архиве документ на части, каждая из которых содержит по одной законченной мысли, и каждую из них сохранить в файле формата htm).

HTML He1p Workshop позволяет работать с готовым документом в формате htm или создавать новые документы с помощью встроенного редактора Notepad и простого графического редактора Image Editor. После компиляции получается компактный файл в формате chm, который содержит все тексты, гиперссылки и рисунки из исходного файла. Кроме создания архива документов HTML He1p Workshop позволяет включить в архив поисковую систему (Search), реализовать работу с наиболее часто встречающимися документами ("избранное" - Favorites), украсить архив наглядными и удобными средствами навигации (кнопками).

Приведём пример создания компактного файла, содержащего описание отчёта. Инсталлируем программу HTML Help. Имена файлов и названия папок, в которых они хранятся, должны содержать не более 8 символов латиницы. Кириллица и длинные имена не допускаются.

Выбрав в меню File > New, в возникшем окне New отметить строку Project и нажать на OK:

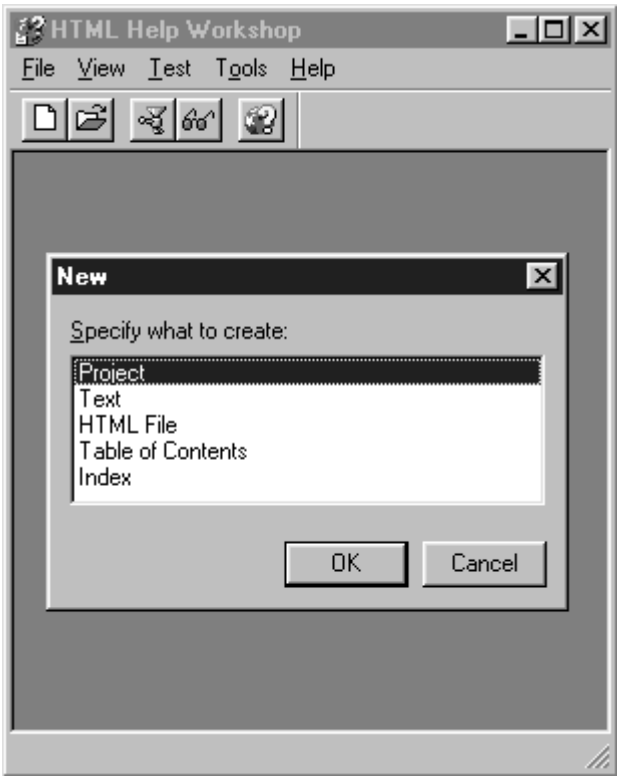


Рис. 4. Окно HTML He1p Workshop.

Появляется окно "волшебника" (Wizard), который рассказывает о возможностях программы. Нажав Next, предстоит указать имя документа на английском языке. В нашем случае – Otchet.

Рис. 5. Окно Wizard.

Затем необходимо выбрать файлы, которые будут использованы в архиве. Отметить нужно HTML - файлы: выбрав нужную папку с заранее подготовленными файлами, включить их список. Выполненную работу нужно сохранить.

Для компиляции файлов нажимаем на кнопку Compile HTML file  . После компиляции выдаётся сообщение об обнаруженных ошибках. При наличии ошибок надо их устранить и повторить компиляцию.

С помощью кнопки Contents, надо наполнить содержанием оглавление отчёта:

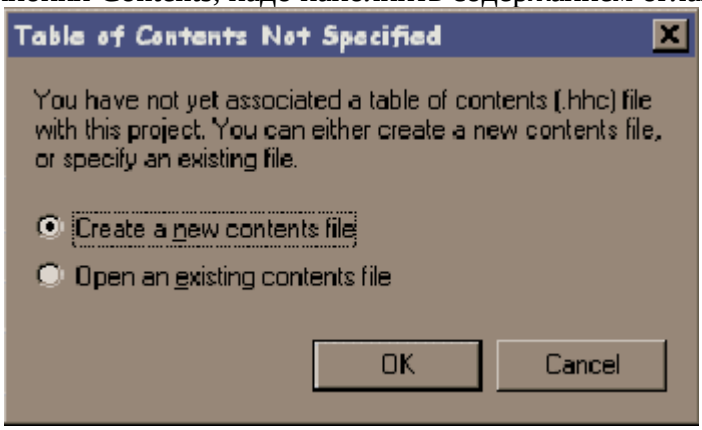


Рис. 6. Окно для запуска системы при создании оглавления.

Основные рабочие кнопки при создании оглавления:

 - Contents properties - вызов таблицы основных свойств (установите поддержку кириллицы, остальное – по усмотрению; General > Font > Change),

 - Insert a heading – установить заголовок документа,

 - Insert a page – установить страницу документа,

 - Edit selection – редактировать отмеченную позицию,

 - Delete selection – удалить отмеченную позицию.

В результате на экране появляется созданное оглавление:

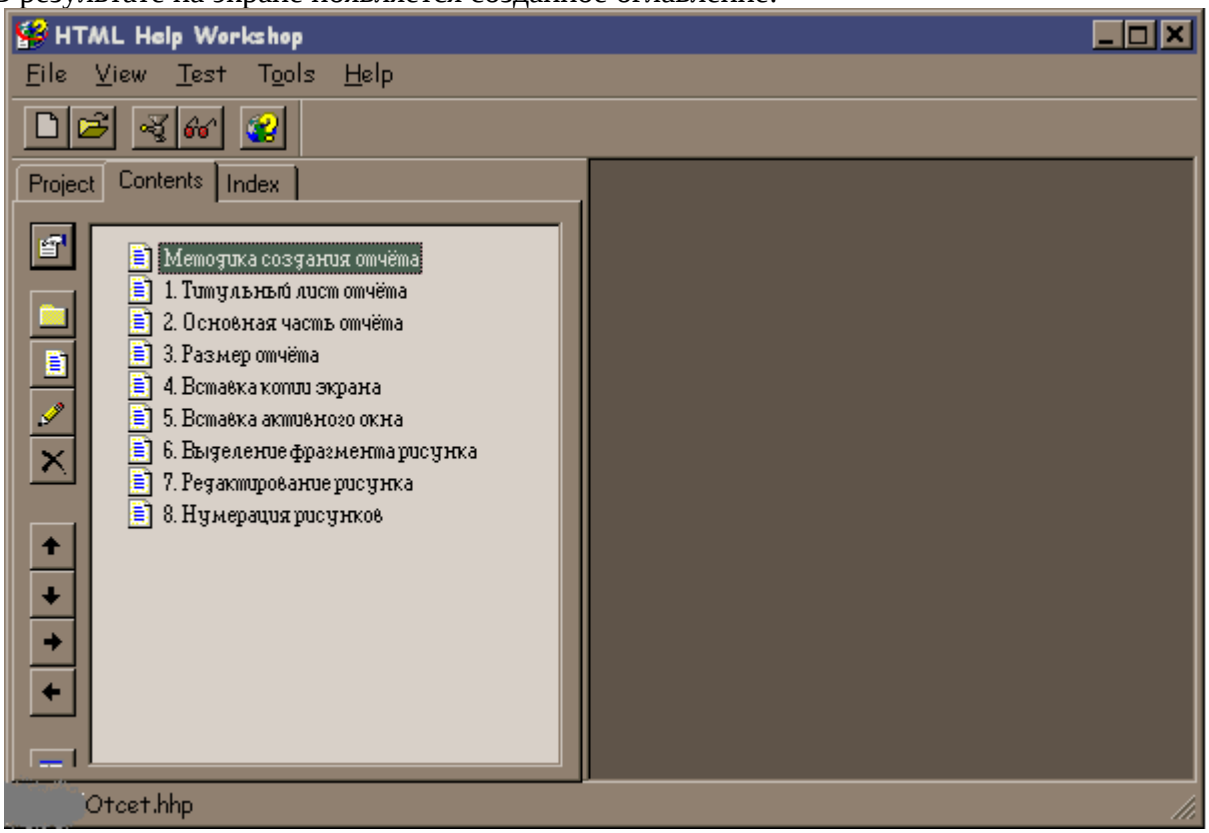


Рис. 7. Экран с созданным оглавлением.

Стрелками (вверх, вниз, вправо, влево) можно перемещать строки, поэтому окончательную сортировку строк оглавления можно произвести в последнюю очередь. При установке заголовка или страницы необходимо дать название ресурсу (Entry Title) и указать его точное месторасположение на компьютере (Edit > Browse). При этом надо иметь в виду, что средства навигации (гиперссылки) не поддерживают некоторые возможности Windows: **русские буквы, длинные имена, пробелы и знаки препинания в адресах содержаться не должны. Все названия папок и файлов должны быть выполнены латиницей и иметь длину не более 8 символов.**

После завершения процедуры создания оглавления необходимо произвести ещё раз компиляцию проекта.

Нажав на клавишу View Compiled File  , вызываем на экран для просмотра созданный архив, уже в формате chm:

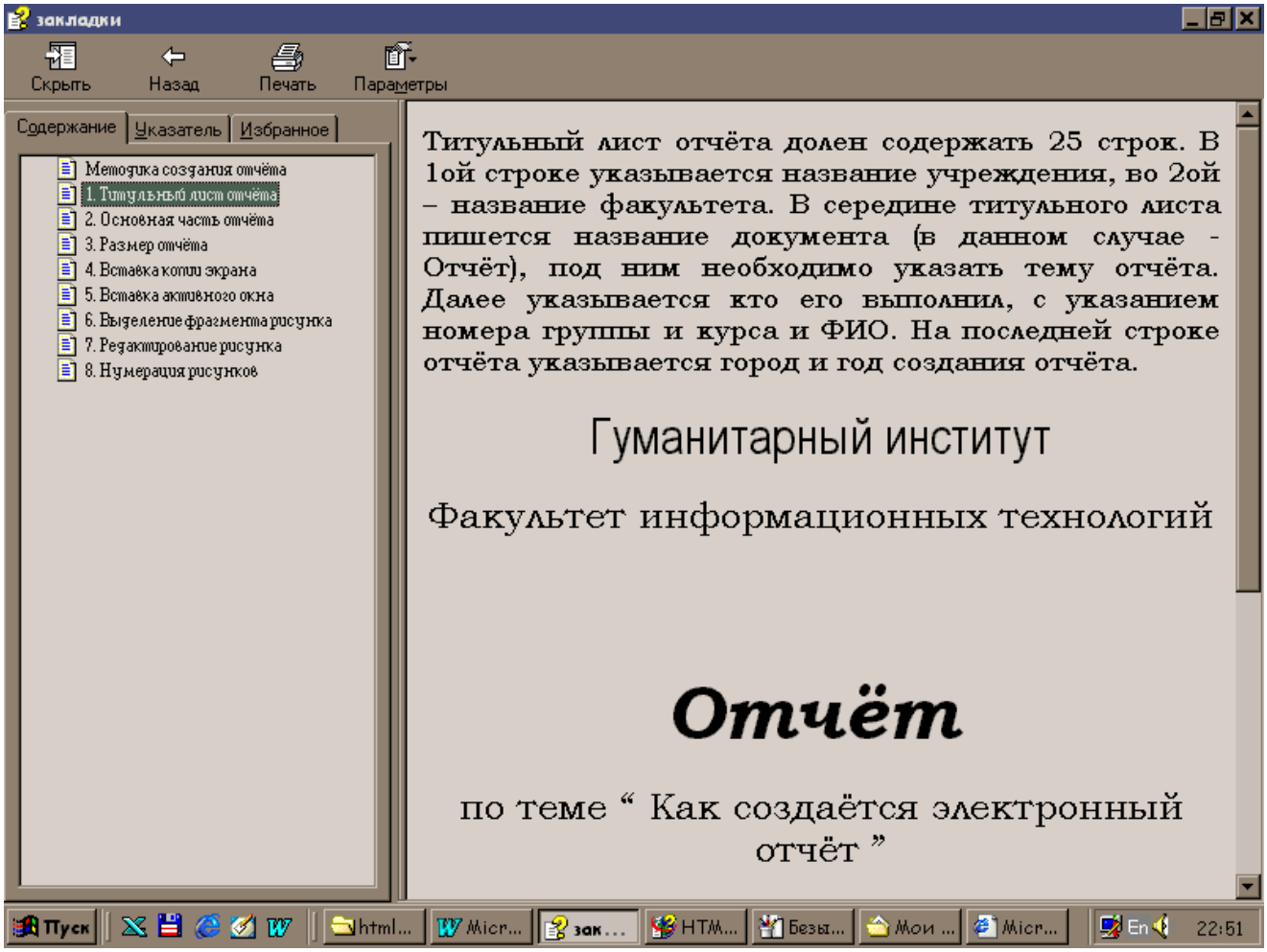


Рис. 8. Архив в формате chm.

После создания оглавления необходимо создать раздел поиска по ключевым словам. Для этого необходимо снова запустить проектный hhp-файл и вызвать меню Index. Если Вы хотите, чтобы поиск можно было производить как по английским, так и по русским ключевым словам, определите с помощью кнопки Index properties базовый язык как русский (Font > Change > Russian).

Список ключевых слов определяет разработчик. Он же указывает путь от ключевого слова к нужному разделу (или разделам).

Основные рабочие кнопки при создании поисковой системы:

 - Insert a keyword - ввести ключевое слово (фразу) (Keyword) и указать путь до нужного раздела (Edit > Browse),

 - Edit selection - редактировать отмеченную позицию,

 - Delete selection - удалить отмеченную позицию,

 - Sort keywords alphabetically - сортировать ключевые слова (фразы) по алфавиту.

При создании индекса весь процесс виден на экране:

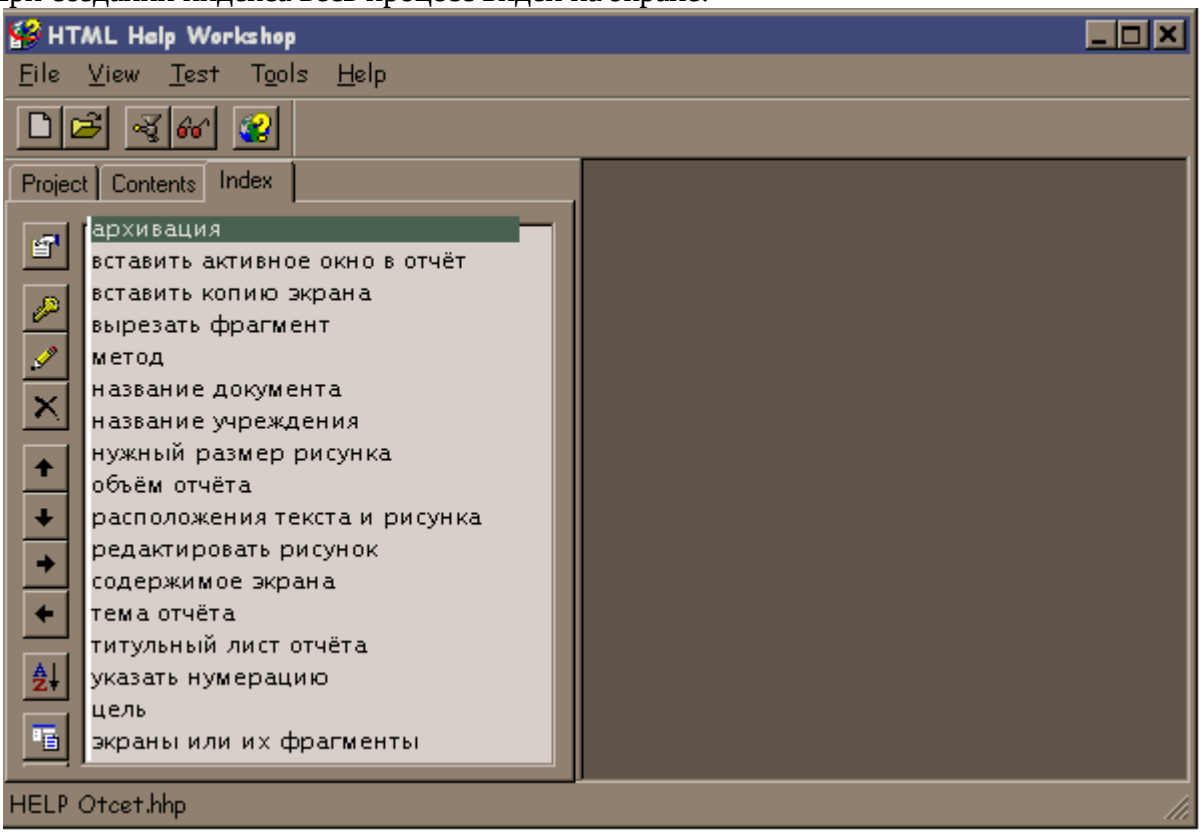


Рис. 9. Экран для создания индекса.

Для генерации системы “полного контекстного поиска”, то есть поиска по всем словам, содержащимся в создаваемом описании нужно запустить проектный hhp-файл, выбрать окно Project, кликнуть кнопку Add/Modify Windows Definitions (Дополнить/Изменить Определения Окон), в открывшейся таблице Window Types (Виды Окна) выбрать меню Navigation Panel (Панель Навигации), отметить галочками окна Search Tab и Advanced, нажать OK. Все дальнейшие шаги вам подскажет Волшебник (Wizard). Не забудьте только отметить Compile full-text information, тогда ваша поисковая система уж точно ничего не пропустит. Теперь, после перекомпиляции проектного файла, в chm-описании добавилось окно Search (Поиск).

В последней версии программы HTML Help появилась возможность добавить на панели меню **Избранное** (**Favorites**). В уже знакомом меню **Navigation Panel**, в строке **Window Type** указать имя, например “Избранное” или “Закладки”, отметить галочкой позицию **Favorites Tab** и нажать на **OK**. Наполнить содержанием и смыслом этот раздел можно, непосредственно редактируя **chm**-файл. Для этого достаточно отметить нужный раздел в окне **Contents** (Содержание), открыть окно **Favorites** (Избранное) и добавить избранный раздел в список излюбленных мест.

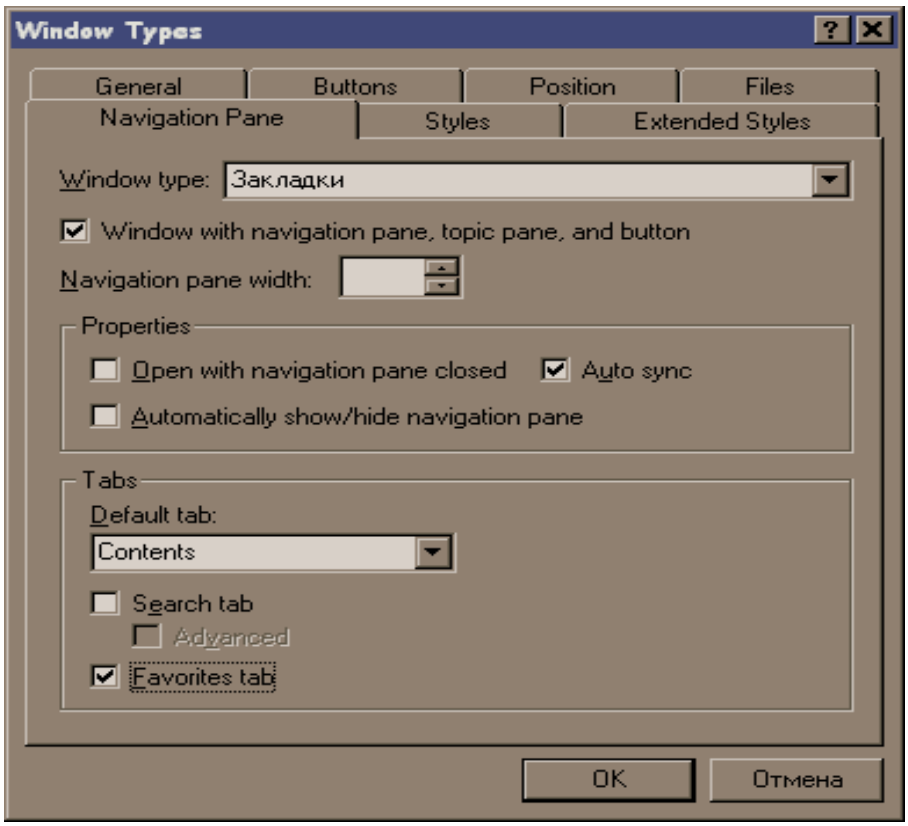


Рис. 10. Настройка пакета на создание меню Favorites.

Для создания дополнительных кнопок:

Кликните на кнопку **Add/Modify Windows Definitions**  , выберите меню **Buttons** (Кнопки):

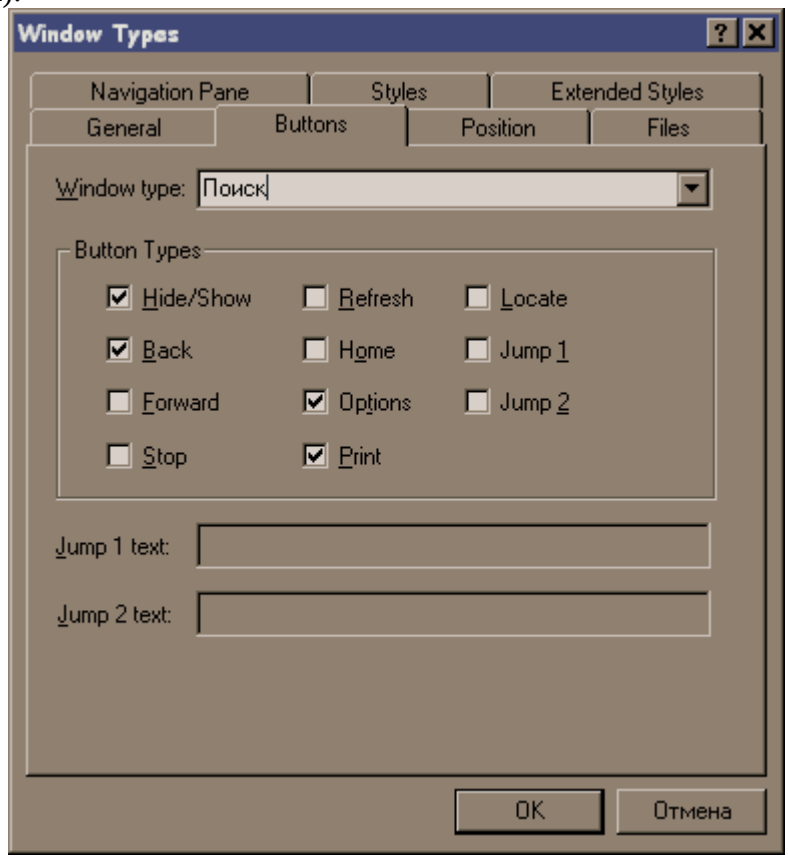


Рис. 11. Настройка пакета на создание дополнительных кнопок.

В меню для настройки пакета на создание дополнительных кнопок можно задействовать следующие опции:
Hide/Show - Скрыть/Показать левый бар
Back - Назад - возврат к предпоследнему шагу
Forward - Вперёд - возврат к последнему шагу
Stop - Остановить загрузку страницы
Refresh - Обновить - перегрузить текущую страницу
Home - Домой - переход на стартовую страницу
Options - Параметры - "все в одном флаконе"
Print - Печать - вывод страницы на печать
Locate - Найти - показ текущей темы в оглавлении
Jump 1/2 - По выбору 1/2 - создание собственной кнопки с переходом на указанную страницу.

После завершения работы нужно снова провести компиляцию.

Инструментарий, облегчающий создание программ

Управление компилятором C#

Консоль команд Microsoft Visual Studio находится по адресу:
Пуск – Все программы – Microsoft Visual Studio - Visual Studio Tools – Visual Studio Command Prompt

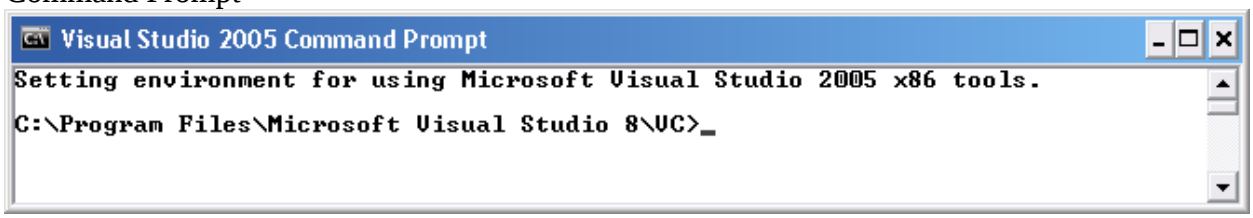


Рис. 12.

При запуске по этому адресу все необходимые переменные окружения уже заданы, консоль настроена для работы с C#.

Все действия в консоли команд лучше выполнять, находясь ближе к корню. Для перехода в корень диска надо выполнить cd \

А для перехода на другой диск (например, d) используется команда d: <Enter>

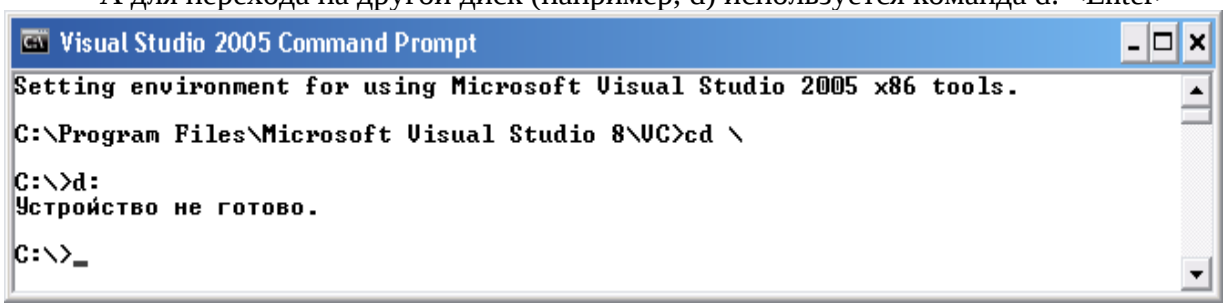


Рис. 13.

Компиляция простейшей программы.
Рассмотрим программу "Hello, world":

```
using System;
class Hello
  {
 static void Main()
  {
  Console.WriteLine("hello, world");
 }
```

}

Поместим эту программу в файл hello.cs и скомпилируем ее командой «csc hello.cs».

Так выглядит вызов компилятора в простейшем случае.

При вызове компилятора нужно убедиться, что правильно выбран путь к csc.exe.

В результате работы компилятора будет получен файл hello.exe.

Кроме исполняемого файла можно также создавать **библиотеки** - dll-сборки, которые могут использоваться другими приложениями, и др.

Возможности компилятора C# (опции командной строки компилятора CSC):

Выходные файлы	
/out:<file>	Имя выходного файла (если имя файла не указано – образуется производное от имени первого указанного cs-файла)
/target:exe	Скомпилировать консольный запускаемый файл (опция по умолчанию) (Краткая форма: /t:exe)
/target:winexe	Скомпилировать запускаемый файл Windows (Краткая форма: /t:winexe)
/target:library	Скомпилировать в библиотеку DLL (Краткая форма: /t:library)
/target:module	Скомпилировать модуль, который может быть добавлен в другую конструкцию (Краткая форма: /t:module) * Примечание: Модуль и библиотека имеют одинаковые расширения (DLL), но несмотря на это, они представляют собой разные конструкции. Библиотека - это классическая библиотека DLL. Модуль - это файл, откомпилированный без создания заголовка (assembly). В дальнейшем модуль можно добавить в любую другую конструкцию при компиляции (с помощью опции /addmodule).
/nooutput	Только проверять ошибки в коде; не создавать запускаемый файл
/define:<список символов>	Определяет директивы препроцессора (Краткая форма: /d) Написание csc /define:mark1 MyApp.cs равносильно наличию в cs-файле строки #define mark1
/doc:<file>	Создать XML файл документации

Входные файлы	
/recurse:<маска>	Включить все файлы, удовлетворяющие маске.
/main:<тип>	Обозначить класс, содержащий точку входа в программу (все остальные будут игнорироваться) (Краткая форма: /m)
/reference:<список файлов>	Включить метаданные из указанных файлов конструкций (Краткая форма: /r) *** Примечание:** все данные, помеченные как public, из указаных файлов конструкций будут включены в данную конструкцию. Если файл не содержит объявления конструкции, то метаданные из него могут быть включены с помощью /addmodule.
/addmodule:<список файлов>	Включить указанные модули в сборку

Ресурсы	
/win32res:<file>	Добавляет файл ресурсов Win32(.res) в выходной файл
/win32icon:<file>	Добавляет иконку в выходной файл
/resource:<resinfo>	Включает .NET ресурс (Краткая форма: /res) * Примечание: resinfo представляет собой следующую последовательность: filename[,identifier[,mimetype]], где filename - имя файла ресурса, identifier - логическое имя ресурса (используется для его загрузки), mimetype - медиа тип ресурса (обычно отсутствует).
/linkresource:<resinfo>	Создает ссылку на ресурсный файл в данной сборке (не включая его в выходной файл)(Краткая форма: /linkres)

Создание кода	
/debug[+|-]	Создавать (не создавать) информацию отладки.
/debug:{full|pdbonly}	Указание типа отладки ('full' - по умолчанию, позволяет прикреплять отладчик к запускаемой программе)
/optimize[+|-]	Запускать оптимизацию (Краткая форма: /o)
/incremental[+|-]	Запускать компиляцию только измененных частей кода(Краткая форма: /incr)
Ошибки и предупреждения	
/warnaserror[+|-]	Относиться к предупреждениям, как к ошибкам
/warn:	Установить уровень предупреждений (0-4) (Краткая форма: /w)
/nowarn:<список предупреждений>	Исключить указанные предупреждения * Примечание: список предупреждений - последовательность их номеров num1[,num2[...]]
Язык	
/checked[+|-]	Выполнять проверки на переполнения и опустошения
/unsafe[+|-]	Допускать unsafe код (unsafe код - небезопасный код)

Разное	
@<file>	Прочтение команд и опций компилятора из файла
/help	Показывает информацию об использовании (Краткая форма: /?)
/nologo	Не показывать копирайт при компиляции
Дополнительно	
/baseaddress:<address>	Базовый адрес для создаваемой библиотеки
/bugreport:<file>	Создавать файл отчета об ошибках
/codepage:<n>	Указать таблицу символов, для использования во время открытия исходного текста
/fullpaths	Компилятор будет указывать полные имена файлов
/nostdlib[+|-]	Не допускать включения стандартной библиотеки (mscorlib.dll). В этой библиотеке хранятся все основные классы .NET Framework.

Примеры команд командной строки для компилятора C#

- Компиляция файла File.cs в файл File.exe:

csc File.cs

- Компиляция файла File.cs в файл File.dll:

csc /target:library File.cs

- Компиляция файла File.cs и создание файла My.exe:

csc /out:My.exe File.cs

- Компиляция всех файлов C# в текущем каталоге с оптимизацией и определением символа DEBUG. Результат File2.exe:

csc /define:DEBUG /optimize /out:File2.exe *.cs

- Компиляция всех файлов C# в текущем каталоге с созданием версии отладки файла File2.dll. Отключение отображения логотипа и предупреждений:

csc /target:library /out:File2.dll /warn:0 /nologo /debug *.cs

- Компиляция всех файлов C# в текущем каталоге в файл Something.xyz (библиотека DLL):

csc /target:library /out:Something.xyz *.cs

Сборка C# - проекта

Следующие примеры взяты с сайта http://www.firststeps.ru.

1. Создание DLL

Для создания DLL на C# нужно просто написать класс без функции Main.

```
using System;

namespace MyClass
{
  public class My
  {
          public static string MyStrimng(string s)
          {
                  return s+"Hello";
          }
  }
}
```

Осталось только правильно скомпилировать:
ключ /target нужен, чтобы указать, как будет компилироваться проект library.
Вот полный пример команды компиляции:

```
csc.exe /target:library /out:Mainw.dll Main.cs
```

В результате компиляции появиться DLL.

Рис. 14.

Обратите внимание: класс должен быть публичный, а методы которые будут доступны должны быть описаны как static. И еще старайтесь в именах не пересекаться стандартными именами. Например использование имени Main для DLL грозит ошибками.

2. Использование DLL

Для использования DLL её нужно для сначала поместить в ту же папку что и файл с кодом.

Рис. 15.

Код напишем:

```
using System;
using System.Windows.Forms;
using MyClass;

class MainClasses
{
  public static void Main(string[] args)
  {
          string s=My.MyStrimng("client ");
          Console.Write(s);
  }
}
```

Достаточно определить пространство имен. Но при компиляции нужно сослаться на используемый DLL для этого есть опция /reference. Вот готовая команда компиляции:

```
csc /out:Main.exe /reference:Mainw.dll Main.cs
```

```
D:\net_step\4>build

D:\net_step\4>csc /out:Main.exe /reference:Mainw.dll Main.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.


D:\net_step\4>main
client Hello
D:\net_step\4>_
```

Рис. 16.

3. Создание модуля и сборки (Assembly)

Если необходимо сделать сборку с другими файлами в dll, мы должны откомпилировать дополнительно подключаемый проект, как модуль. Для этого есть специальный ключ /target:module.

Команда компиляции:

```
csc.exe /target:module  Mainw.cs
```

В результате компиляции мы получим именно модуль с расширением netmodule.

Рис. 17.

Теперь добавим модуль созданного ранее клиентского приложения:

```
csc /addmodule:Mainw.netmodule  /t:module Main.cs
```

Появится новый модуль:

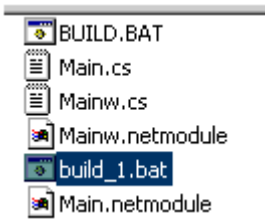


Рис. 18.

И вот теперь мы можем создать сборку воспользовавшись утилитой AL:

```
al Main.netmodule Mainw.netmodule /main:MainClasses.Main
/out:myAssembly.exe /target:exe
```

И вот только теперь, если Все нормально и нет ошибок вы получите сборку.

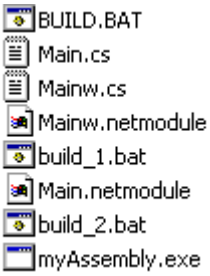


Рис. 19.

Текст созданного ранее клиентского приложения:

```
using System;
using System.Windows.Forms;
using MyClass;

class MainClasses
{
 public static void Main(string[] args)
 {
        string s=My.MyStrimng("client ");
        Console.Write(s);
 }
}
```

4. Global Assembly Cache

- это то место, в котором находятся сборки. Если сборка будет распределена между многими приложениями то она должна быть установлена в это место. Оно находится по пути:

```
:\WINNT\assembly
```

При просмотре его будет использована Shfusion.dll которая отвечает за просмотр.

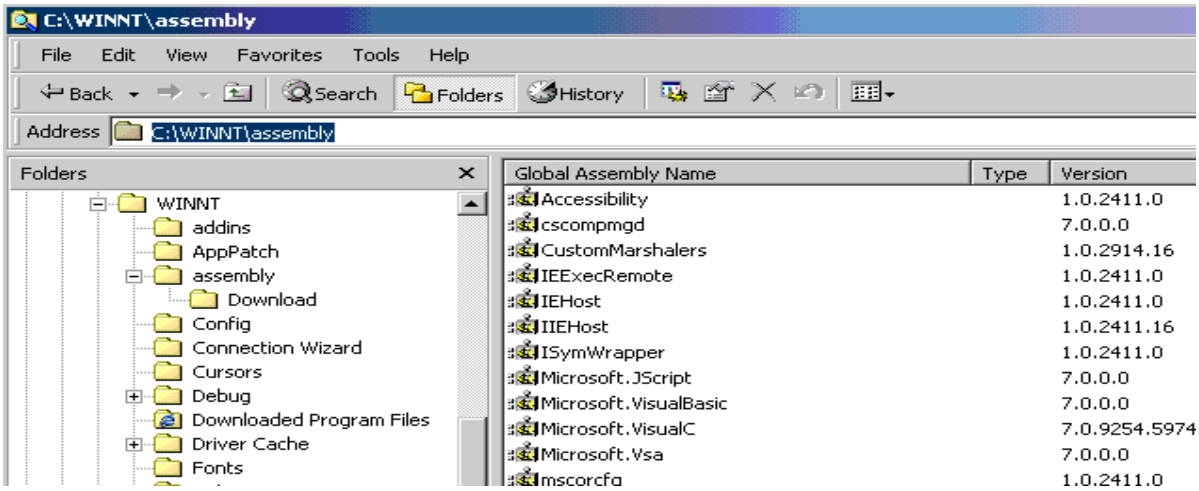


Рис. 20.

Добавлять и удалять сборку из глобального кеша можно с помощью утилиты gacutil:

Добавляем
```
gacutil /i mydll.dll
```
Удаляем
```
gacutil /u mydll
```

5. Использование стандартных DLL

Несмотря на большое количество классов в Net Classes, все равно может потребоваться использовать старое DLL. Например, если Вы переписываете проект и у Вас есть возможность использовать старые DLL, кроме того вообще часть функций реализована в виде DLL и наверно дальше будет реализовываться.

Создание DLL на C# очень сильно упрощено что говорит о том что они будут использоваться дальше. Смотрим код.

```
using System;
using System.Runtime.InteropServices;

class MainClass
{

[DllImport("kernel32")]
public static extern bool  Beep(int _Int1, int _Int2);
```

```
public static void Main(string[] args)
{
        Beep(300,100);
}
}
```

Мы использовали DLLImport указав имя библиотеки и после нее описали функцию, которую будем использовать, описывать ее нужно с правильными параметрами а для этого посмотреть WIN32 API. Данная программа издаст звук.

Основные принципы использования ресурсов.

Практически каждое программное приложение использует ресурсы. К ресурсам относятся любые неисполняемые данные, которые логически развертываются вместе с приложением. Ресурсы могут отображаться в приложении в виде сообщений об ошибках либо как часть интерфейса пользователя. В ресурсы можно импортировать все что угодно, любой файл: можно **WAV**, можно **HTML**, можно **MDB, EXE** и так далее.

В MSDN принято считать, что основным для ресурсов приложения является возможность настроить их на определенный язык и региональные параметры, что позволяет создавать локализованные (переведенные на соответствующий язык) версии приложений. Приложение загружает соответствующие локализованные ресурсы на основе значения свойства CultureInfo.CurrentUICulture, определяющего на какой именно язык приложение должно быть настроено. Это значение устанавливается либо явным образом в коде приложения, либо с помощью общей среды исполнения, исходя из языковых настроек текущего пользователя на локальном компьютере.

Фактически же использование ресурсов не ограничивается настройкой их на определенный язык и региональные параметры. Они могут содержать не только разноязыковые версии приложений, но и данные различных видов, включая символьные строки, изображения, звуковые файлы и объекты.

Ресурсы приложения можно развернуть во вспомогательных (сопутствующих) сборках, содержащих только файлы ресурсов (в них отсутствует код приложения). В такой модели развертывания ресурсов можно создать приложение с одной стандартной сборкой (которая является основной) и несколькими вспомогательными сборками. Например, в основной сборке содержится код программы тестирования знаний, а наборы тестов по разным темам или предметам размещаются во вспомогательных сборках.

Поскольку вспомогательные сборки не являются частью основной сборки, можно легко заменять или обновлять ресурсы, относящиеся к определенному языку, региональным параметрам и другим специфическим блокам информации, не заменяя основную сборку приложения. Данные можно сохранять в файле ресурсов и затем изменять их, не компилируя все приложение заново.

Проекты Visual Studio предоставляют два варианта обработки ресурсов: ресурсы могут быть связаны – linked (этот вариант используется по умолчанию) или внедрены - embeded. В одном проекте можно иметь и связанные и внедренные ресурсы. Однако, чаще всего для всех ресурсов в проекте выбирается какой-либо один вариант.

Связанные ресурсы хранятся в виде файлов в проекте; во время компиляции, данные ресурсов берутся из файлов и добавляются к манифесту приложения. Файл ресурсов приложения (.resx) хранит только относительный путь или ссылку на файл, хранящийся на диске.

В случае внедренных ресурсов, данные ресурсов хранятся непосредственно в файле .resx в текстовом представлении двоичных данных.

В любом случае, данные ресурсов компилируются в исполняемый файл.

Связанные ресурсы можно изменить на внедренные, путем изменения свойства Persistence для файла ресурсов.

В MSDN считается, что в общем случае следует использовать связанные ресурсы, поскольку они являются простыми в использовании - их можно редактировать непосредственно внутри проекта и легко добавить или удалить по мере необходимости.

Тем не менее, имеются случаи, в которых внедренные ресурсы могут стать лучшим выбором. Внедренные ресурсы являются наилучшим решением, если необходимо совместно использовать файлы ресурсов (.resx) приложения в нескольких проектах. Например, при наличии общего файла ресурсов с логотипами организации, сведениями о товарном знаке и т.п., использование внедренных ресурсов означает, что достаточно скопировать только файл .resx и не сопоставлять файлы данных ресурсов.

Внедренные ресурсы нельзя редактировать непосредственно. При попытке изменить внедренный ресурс, появится сообщение, предлагающее преобразовать элемент в связанный ресурс, чтобы его отредактировать. Преобразование рекомендуется, но необязательно. Ресурсы необходимо экспортировать, внести изменения во внешней программе и затем импортировать обратно в проект.

При добавлении ресурсов в проект Visual Studio, они создаются как связанные ресурсы. В некоторых случаях, возможно, потребуется внедрить ресурс в файл ресурсов (RESX) приложения.

Для правки ресурсов проекта можно использовать Конструктор ресурсов. К поддерживаемым Конструктором ресурсов типам ресурсов относятся строки, изображения, значки, звук и файлы.

Связанные ресурсы могут быть изменены на внедренные на странице Ресурсы в окне Конструктор проектов.

Каждый тип ресурсов редактируется с использованием редактора, связанного с расширением имени файла ресурса. Например, редактором по умолчанию для растровых изображений (расширение BMP) может быть редактор ресурсов Visual Studio, а редактором по умолчанию для JPEG изображений (расширение JPG) может быть Windows Paint. Любой файл ресурсов можно открыть в другом редакторе, используя команду меню Открыть с помощью, а также можно изменить редактор по умолчанию для любого типа ресурсов.

Конструктор ресурсов не зависит от выбранного языка и поддерживает проекты, разработанные на всех языках Visual Studio. Для Visual C# Конструктор ресурсов создает строго типизированные ресурсы в пространстве имен проекта.

Для работы с ресурсами в Visual Studio используются различные средства.

Файл, содержащий строковые ресурсы, создаётся в виде текстового файла.

С помощью Генератора файлов ресурсов (Resgen.exe) производится преобразование текстового файла в двоичный файл ресурсов (.resources).

После образования двоичного файла ресурсов можно

- внедрить двоичный файл ресурса в исполняемый файл приложения (с расширением exe) или
- в библиотеку приложений (с расширением dll) с помощью компилятора языка csc, или
- внедрить его в сопутствующую сборку с помощью Компоновщика сборок (Al.exe) .

Файл ресурсов, содержащий строки, изображения или данные объекта, может быть создан типа XML (с расширением .resx). Для преобразования файла .resx в двоичный файл ресурсов (.resources) можно использовать Генератор файлов ресурсов (Resgen.exe). XML-файл ресурсов (RESX) может быть создан программным путём с помощью типов пространства имен System.Resources.

Строковые ресурсы можно вырезать, копировать и вставлять между строками в редакторе строк; однако невозможно вставлять отдельные значения или комментарии.

При копировании и вставке строк из внешнего источника, поля Имя, Значение и Комментарий должны быть разделены табуляцией. В противном случае, вся строка добавляется к столбцу Имя.

Ресурсы в текстовом файле могут быть созданы только построчно.

Текстовый файл, содержащий строковые ресурсы, должен иметь следующий формат:

# This is an optional comment.
name = value
; This is another optional comment.
name = value

Строковые ресурсы представляются в виде пар: ключ – значение.

Комментарии отмечаются точкой с запятой или решёткой в начале строки.

Строки, содержащие комментарии, размещаются отдельно. Они не включаются в компилируемый с помощью Resgen.exe файл ресурсов.

Пустые строки в файле ресурсов игнорируются.

Следующий пример показывает исходный код консольного приложения, которое использует .resources-файл для вывода сообщения пользователю

```
using System;
using System.Reflection;
using System.Resources;

public class Example
{
   public static void Main()
   {
      ResourceManager rm = new ResourceManager("GreetingResources",
                               Assembly.GetExecutingAssembly());
      Console.Write(rm.GetString("prompt"));
      string name = Console.ReadLine();
      Console.WriteLine(rm.GetString("greeting"), name);
   }
}
// The example displays output like the following:
//       Enter your name: Wilberforce
//       Hello, Wilberforce!
```

Если файл с исходным кодом имеет имя Greeting.cs, ресурсный файл называется GreetingResources.resources, включение ресурсного файла в программу производится по следующей команде:

csc greeting.cs /resource:GreetingResources.resources

Ресурсы в Resx – файле.

Файл XML ресурсов (.resx) в отличие от текстовых ресурсов может содержать и строки, и двоичные данные (например, графические образы), иконки, аудио клипы или программные объекты.

Этот тип ресурсного файла содержит стандартный XML-заголовок.

Все данные, содержащие пары ключ-значение заключены в тэги данных.

Например, следующий тэг данных содержит строку ресурса, именуемую приглашением: «Введите ваше имя»:

<data name="prompt" xml:space="preserve">
<value>Enter your name:</value>
</data>

RESX-файл нельзя внедрить в исполняемый файл или скомпилировать во вспомогательную сборку. Вначале необходимо преобразовать файл .resx в двоичный файл ресурсов (.resources) с помощью Генератора файлов ресурсов (Resgen.exe) .

Ресурсы в файле типа .resources.

Для создания двоичного ресурсного (.resources) файла можно воспользоваться классом System.Resources.ResourceWriter для получения ресурса прямо из кода, или воспользоваться Resource File Generator (Resgen.exe) для создания ресурсного файла из текстового или .resx-файла.

Ресурсный файл этого типа может содержать двоичные данные (массив байтов) или данные в виде объектов, добавленные к строковым данным.

Программное создание .resources-файла предусматривает следующие действия:

1. **Создание объекта класса ResourceWriter с уникальным именем файла. Это можно сделать с помощью конструктора данного класса.**
2. **Вызвать один из перегруженных методов ResourceWriter.AddResource для каждого ресурса, загружаемого в файл. Ресурс может быть строкой, объектом или коллекцией двоичных данных (массивом байтов).**
3. **Вызов метода ResourceWriter.Close для записи ресурсов в файл и закрытия объекта ResourceWriter.**

После создания файла .resources, его можно внедрить в исполняемый файл во время выполнения или в библиотеку, с помощью команды /resource компилятора csc, или встроить в сопутствующую сборку с помощью Компоновщика сборок (Al.exe) .

Создание файлов ресурсов

Библиотека базовых классов .NET Framework содержит несколько классов в пространстве имен System.Resources , применяемых при создании ресурсов и работе с ними в приложениях.

Файлы ресурсов можно создавать тремя способами. Если ресурсы содержат только строковые данные, самый простой метод — вручную создать текстовый файл. Если ресурсы содержат объекты или сочетание строк и объектов, следует создать файл .resx или файл .resources.

Для записи объектов в файл ресурсов необходимо, чтобы объекты были сериализуемыми.

Класс	Описание
Интерфейс IResourceReader	Считывает ресурсы из потока.
Интерфейс IResourceWriter	Записывает ресурсы в поток.
Класс ResourceReader	Считывает ресурсы в двоичном формате файла ресурсов, используемом средой выполнения по умолчанию. Этот класс — стандартная реализация интерфейса **IResourceReader**.
Класс ResourceWriter	Записывает ресурсы в двоичном формате файла ресурсов, используемом средой выполнения по умолчанию. Этот класс — стандартная реализация интерфейса **IResourceWriter**.
Класс ResXResourceReader	Предоставляет возможность извлекать ресурсы из файлов .resx. Это специализированная реализация интерфейса **IResourceReader**.
Класс ResxResourceWriter	Предоставляет возможность создавать файлы .resx из указанных ресурсов. Это специализированная реализация интерфейса **IResourceWriter**. Для преобразования файлов .resx в формат .resources следует использовать генератор файлов ресурсов (Resgen.exe).

Работа с ресурсами.

Для получения навыков работы с ресурсами выполните следующую последовательность действий, описанных в сайте www.firststep.ru: создание файла с ресурсами строк; извлечение ресурсов из файла; присоединение файла ресурсов к exe–модулю программы; размещение картинки в ресурсах; отображение картинки из ресурсов.

Создаем файл с ресурсами строк

Файл с ресурсами строк должен иметь расширение **TXT**. В нем могут находиться только строки, он имеет следующий формат:

```
[header]          заголовок
;comments         комментарии
name = value      значения
```

Давайте создадим тестовый файл с именем resource.txt и текст в нем.

```
string1=Hello resouce file
string2=My Resource File
```

А теперь нам нужно воспользоваться утилитой **resgen** для получения файла ресурсов. Создадим файл с именем**buildresoure.bat** и код в нем.

```
Resgen.exe resource.txt
```

Все запускаем на выполнение.

В результате будет сформирован файл **resource.resources**.

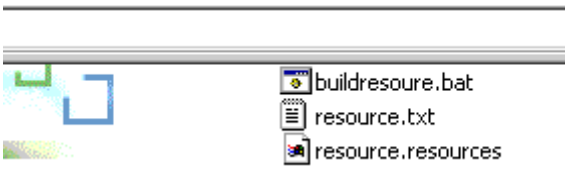


Рис. 21.

Извлекаем ресурсы из файла

Поместим файл resource.resources в папку с проектом.

Напишем код.

```
using System;
using System.Resources;
class MainApp
{
    public static void Main()
    {
        ResourceManager rm =
ResourceManager.CreateFileBasedResourceManager("resource",".",null);
         Console.WriteLine(rm.GetString("string1"));
         Console.WriteLine(rm.GetString("string2"));
    }
}
```

Для того, чтобы работать с ресурсами нам нужно пространство имен **System.Resources** там находятся классы для работы с ресурсами. Один из этих классов **ResourceManager** он отвечает за загрузку ресурсов и управление ими. Мы загрузили ресурсы из файла, и вывели строки на экран.

```
Command Prompt
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\>d:

D:\>cd net_step

D:\net_step>cd 3

D:\net_step\3>3
Hello resouce file
My Resource File

D:\net_step\3>
```

Рис.22.

Присоединяем файл ресурсов к exe – модулю программы

Как пользоваться внешним файлом мы посмотрели, теперь будет смотреть как прикрепить файл с ресурсами к программе. Файл должен быть уже скомпилирован в смысле файл ресурсов.

Соединения файла **EXE** и ресурсов производиться при компиляции. Это делается путем использованию ключа **/res** при компиляции. Вот пример командной строки для компиляции:

```
csc /res:resource.resources 4.cs
```

А вот код программы:

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.IO;
using System.Resources;


class MyForm : AppForm
{
  public static void Main()
  {
          Application.Run(new MyForm());
  }
}


class AppForm : Form
{
  public AppForm()
  {
          rm = new ResourceManager("resource",this.GetType().Assembly);
          MessageBox.Show(rm.GetString("string2"));
  }
  private ResourceManager rm;
}
```

Здесь мы воспользовались классом **ResourceManager** который умеет работать с ресурсами, загружать их, получать ресурс по названию. Он находиться в пространстве имен **using System.Resources;**. Воспользовались функций **GetString** для получения строки из ресурсов. Итак компилируем и запускаем. У нас появиться окно перед запуском формы в котором будет строка из файла ресурса.

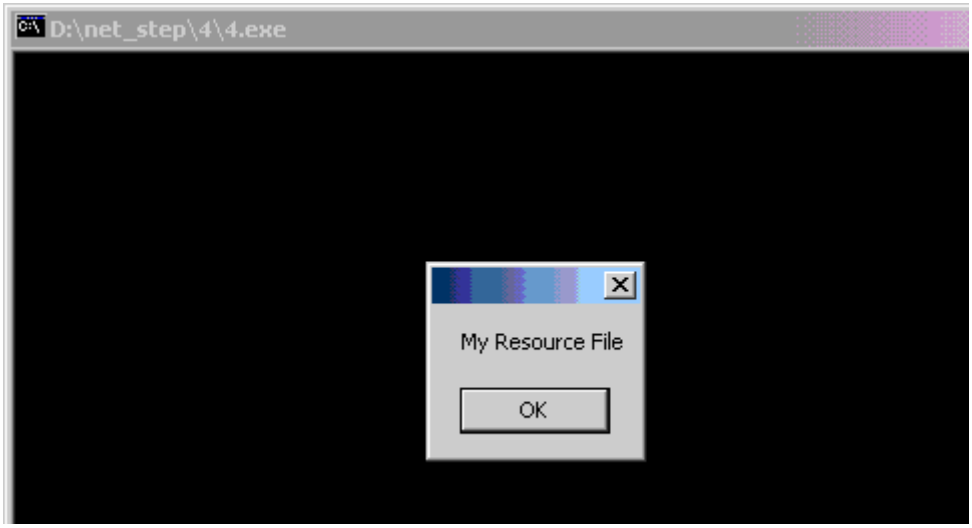


Рис.23.

Первоначальный текстовый файл был такой:

```
string1=Hello resouce file
string2=My Resource File
```

Как поместить картинку в ресурсы

Перевод картинки в ресурсы выполним с помощью программы resgen.

Используя ее можно перевести файл **bmp** или **jpeg** в формат **resx**. Это файл в формате XML с которым программа сможет работать. Итак поместим в папку **BMP** файл, **ResXGen** и создадим сборку:

```
ResXGen /i:1.bmp /o:Images.resx /n:Image
```

Здесь ключи.

```
/i входной файл
/o выходной файл
/n имя ресурса
```

Запустим ResXgen.

Результат будет помещён в файл в формате **resx**, то есть **XML**.

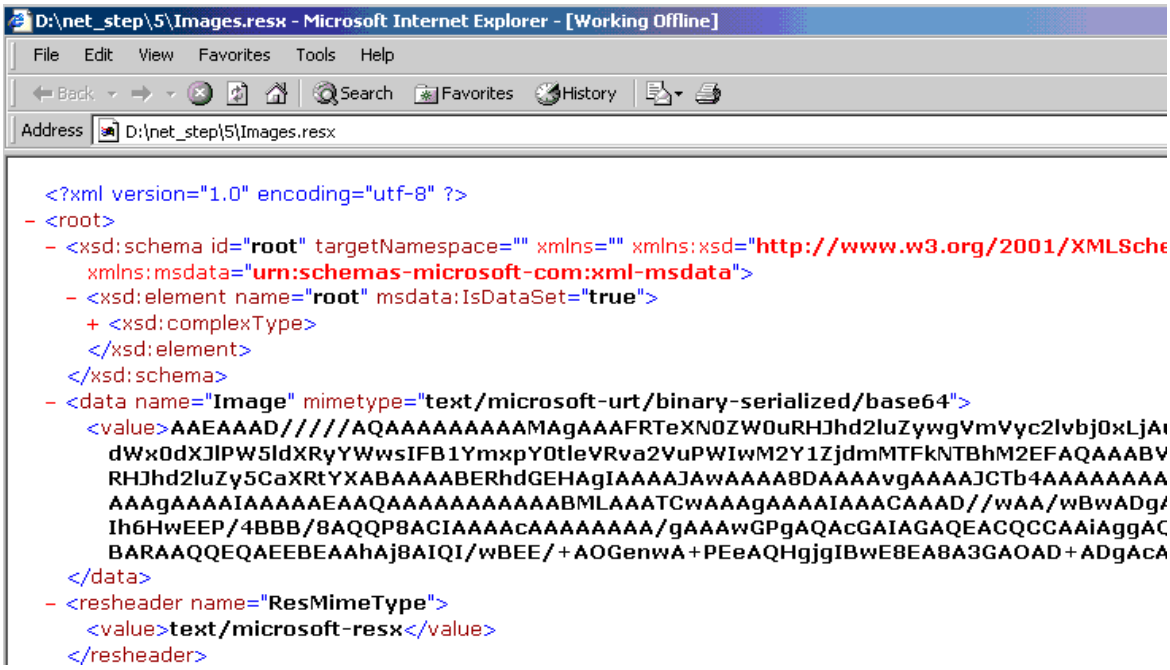


Рис.24.

Отображаем картинку из ресурсов

Если картинка помещена в файл в формате **resx, п**режде чем её присоединить к проекту, нужно ее перевести в файл **Resource**. Это можно сделать с помощью известной утилиты **Resgen:**

```
Resgen.exe Images.resx
```

Потом при сборке присоединить её к **EXE**

```
csc /res:Images.resources 6.cs
```

Потом только можно пользоваться.

Вот код.

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.IO;
using System.Resources;

class MyForm : AppForm
{
  public static void Main()
  {
          Application.Run(new MyForm());
  }
}


class AppForm : Form
{
  public AppForm()
  {
          MainMenu mnuFileMenu = new MainMenu();
          this.Menu = mnuFileMenu;
          MenuItem MenuItemFile = new MenuItem("&File");
          MenuItemFile.MenuItems.Add("Open",new
System.EventHandler(this.MenuOpen_Click));
          mnuFileMenu.MenuItems.Add(MenuItemFile);
          lb= new Label();
          lb.Location = new Point(15,15);
          lb.Size = new Size(200,200);
          this.Controls.Add(lb);
  }
  private void MenuOpen_Click(Object sender, EventArgs e)
  {
          ResourceManager rm = new
ResourceManager("Images",this.GetType().Assembly);
          lb.Image= (System.Drawing.Image)rm.GetObject("Image");
  }
  private Label lb;
}
```

При нажатии на **Open** картинка будет отображена.

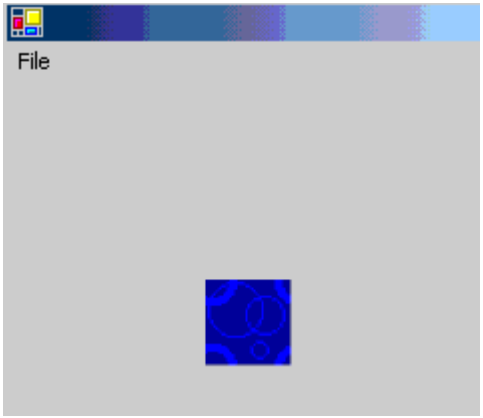


Рис. 25.

Программирование с использованием компонентов

Основные определения.

Компонентный подход к разработке программного обеспечения характеризуется тем, что создаваемый программный продукт состоит из взаимодействующих компонентов.

При этом различные компоненты могут независимо разрабатываться разными группами программистов, и

при создании каждого компонента может применяться наиболее подходящий язык программирования.

Определение термина **компонент**:

- Компонент представляет собой отдельный модуль. Функциями компонента может воспользоваться любое приложение, но сам компонент не является автономной программой.
- Компонент определён, как независимый модуль.
- Компонент предназначен для многократного применения.
- Компонент должен быть представлен в двоичном формате. Использовать компонент могут многие клиенты, но доступа к его исходному коду они не имеют.
- Программа, которая использует компонент, называется *клиентом* (client).
- Компонент может работать с любым количеством клиентов.

Для того, чтобы клиент мог использовать компонент, необходимо, чтобы и клиент, и компонент подчинялись одному и тому же набору правил.

- Набор правил, определяющих форму и поведение компонента, называется компонентной моделью (component model).
- Компонентная модель в среде .NET Framework определяется

интерфейсом System.ComponentModel.IComponent и
классом System.ComponentModel.Component.
Этот класс обеспечивает стандартную реализацию интерфейса
IComponent по умолчанию.

- Если класс наследует класс Component, он автоматически выполняет правила, необходимые для получения .NET-совместимого компонента.

Для примера: Свойство CanRaiseEvents этого класса возвращает значение, показывающее, может ли компонент вызывать событие, а свойство Events возвращает список обработчиков событий, которые прикреплены к этому объекту Component.

В C# с компонентами связаны две другие конструкции: *контейнеры* и *узлы*.

- **Контейнер** – это группа компонентов. Контейнеры упрощают программы, в которых используется множество компонент.

- **Узел** позволяет связывать компоненты и контейнеры. Узел идентифицирует компонент и согласно конструкции интерфейса IComponent, сообщает ему факт отсутствия компонента в контейнере, если такой имеет место.

В компонентной системе можно выделить три вида взаимодействия компонентов:

- взаимодействие внутри адресного пространства одного процесса;
- межпроцессное взаимодействие, при котором компоненты работают в разных процессах;
- взаимодействие в сети, когда компоненты запущены на разных компьютерах.

Межпроцессное и сетевое взаимодействие используется, главным образом, в распределенных системах, изучение которых выходит за рамки данной дисциплины.

Компонентные технологии могут иметь различные виды:

- Библиотеки стандартных подпрограмм;
- Динамические библиотеки подпрограмм;
- Открытые исходные тексты;
- Технологии COM (Component Object Model) и
- Технологии CORBA (Common Object Request Broker Architecture).
- Технология .NET

Библиотеки стандартных подпрограмм.

Наиболее древний способ заключается в использовании библиотек подпрограмм.

Такие библиотеки создаются одной группой разработчиков, а затем могут быть использованы другими программистами в других проектах.

Исходный код библиотек может быть закрыт, то есть они могут распространяться в откомпилированном виде, защищая тем самым интересы разработчиков.

Организация компонентного программирования на базе библиотек подпрограмм обладает следующими недостатками:

- Двоичный код библиотеки жестко привязан к аппаратной платформе, так как содержит инструкции конкретного процессора. Это уменьшает переносимость программы, использующей библиотеку.
- Как правило, библиотека рассчитана на использование конкретного компоновщика и может поддерживать ограниченное количество языков программирования, компиляторы которых генерируют объектные файлы в нужном формате и используют нужные соглашения о вызове подпрограмм.
- Библиотеки подпрограмм плохо подходят для объектно-ориентированных языков, так как не содержат никакой информации о типах.

Динамические библиотеки подпрограмм.

Динамические библиотеки некоторым образом облегчают компонентное программирование.

- Они поддерживаются операционной системой, и поэтому разработчики компиляторов вынуждены следовать требованиям, налагаемым операционной системой.
- То есть динамические библиотеки не зависят от причуд компиляторов и компоновщиков и могут использоваться для большего диапазона языков программирования.
- Однако в остальном они не лучше, чем обычные библиотеки подпрограмм.

Открытые исходные тексты

В мире открытых исходников компоненты распространяются в виде исходных текстов.

Тем самым вроде бы решается часть проблем.

По крайней мере, можно попытаться переносить эти компоненты с платформы на платформу (перекомпилируя исходники и исправляя возникающие ошибки).

- Исходные тексты в отличие от двоичных файлов содержат информацию о типах, что позволяет использовать объектно-ориентированные возможности.

Однако,

- Если компоненты написаны на разных языках, то соединить их вместе не так-то просто.

Отсюда проистекает стремление сторонников открытых исходников к превращению своих программ в набор маленьких утилит, написанных на C и соединенных посредством скриптов.

Открытые компоненты, например, выполненные в GNU Public License – GPL, требуют, чтобы программы, использующие эти компоненты, сами распространялись в виде открытых исходников.

Это существенно ограничивает применимость открытых компонентов.

Технологии COM и CORBA

Особого внимания заслуживают технологии COM (Component Object Model) и CORBA (Common Object Request Broker Architecture).

- Технология COM, разработанная корпорацией Microsoft, поддерживает все три вида взаимодействия компонентов, а именно:
  - взаимодействие внутри адресного пространства процесса,
  - межпроцессное взаимодействие и
  - взаимодействие по сети.
- Технология CORBA ориентирована исключительно на взаимодействие компонентов по сети.

В COM и CORBA необходимо описывать метаданные на языке IDL (Interface Definition Language).

Подобное неудобство объясняется тем, что эти технологии не подразумевают специальной поддержки в компиляторах языков программирования.

Например, можно написать COM-объект на языке C++, но если компилятор C++ ничего не знает о COM, он не сгенерирует никаких метаданных!

Вот поэтому метаданные во многих случаях нужно отдельно описывать на языке IDL.

Технологии COM и CORBA определяют двоичный стандарт для передачи данных между компонентами.

Так как компоненты могут быть написаны на разных языках и, соответственно, внутреннее представление данных в них может существенно различаться,

- то компонент, осуществляющий передачу, вынужден выполнять преобразование передаваемых данных в стандартное представление (**маршалинг**),
- а компоненту, принимающему данные, приходится выполнять преобразование данных из стандартного представления к своему внутреннему представлению (**демаршалинг**).

При межпроцессном взаимодействии и взаимодействии по сети затраты на преобразование данных не играют существенной роли, но при взаимодействии внутри адресного пространства одного процесса они могут сильно сказаться на производительности программы.

На рисунке изображена схема взаимодействия двух объектов при использовании технологии COM или CORBA:

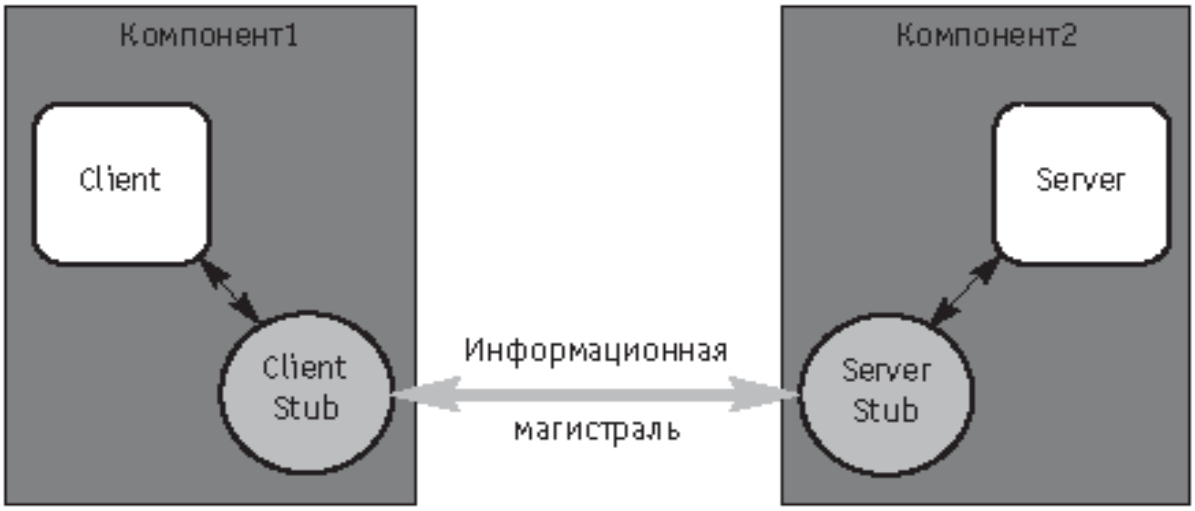


Рис. 26.

Объекты Client и Server находятся в разных компонентах, поэтому объект Client не может непосредственно передать сообщение объекту Server.

Вместо этого он обращается к специальному объекту-заглушке ClientStub, который осуществляет упаковку сообщения и затем передает его информационной магистрали (в терминах CORBA информационная магистраль называется ORB - Object Request Broker).

Информационная магистраль передает сообщение серверному объекту-заглушке ServerStub, который распаковывает сообщение и вызывает соответствующий метод объекта Server.

Результат, возвращаемый методом объекта Server, совершает обратный путь к объекту Client аналогичным образом.

Данный пример показывает, что использование технологий COM и CORBA связано с большими трудностями.

Объекты-заглушки могут быть сгенерированы автоматически на основе описания объекта Server на языке IDL (для этого существуют специальные компиляторы).

Хотя технологии COM и CORBA продолжают активно применяться, есть все основания утверждать, что в самом ближайшем будущем они будут вытеснены более эффективными и удобными технологиями (например, .NET).

Технология .NET.

Организация компонентного программирования на платформе .NET необычайно проста и удобна для программиста.

- Во-первых, компилятор любого языка программирования, реализованного на платформе .NET, генерирует сборки, содержащие CIL-код и метаданные. При этом формат этих сборок определяется спецификацией CLI и не зависит от языка программирования.
- Во-вторых, любая программа, работающая в среде .NET, использует общую систему типов CTS. Поэтому представление данных в памяти определяется CTS и не зависит от языка программирования, на котором написана программа.
- В-третьих, выполнение любой программы управляется средой выполнения CLR.

Это означает, что среда выполнения

- контролирует JIT-компиляцию CIL-кода программы,
- выполняет управление памятью и
- в каждый момент времени имеет всю информацию о состоянии программы.

Эти три особенности платформы .NET позволяют среде выполнения автоматически обеспечивать взаимодействие компонентов вне зависимости от того, на каком языке они написаны.

Взаимодействие компонентов в среде .NET

На рисунке изображена схема взаимодействия двух объектов на платформе .NET.

- Объекты Client и Server находятся в разных компонентах, работающих в адресном пространстве одного процесса.
- Передача сообщения от объекта Client объекту Server сводится к простому вызову соответствующего метода объекта Server, то есть в среде .NET не нужны никакие объекты-заглушки.

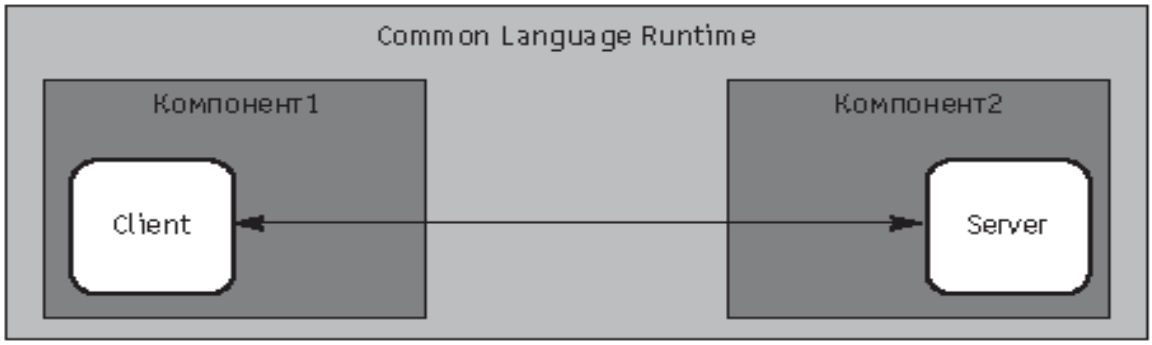


Рис. 27.

Компоненты на платформе .NET представляют собой сборки .NET.

Сборка .NET может статически импортировать любую другую сборку и свободно использовать типы, экспортируемые из этой сборки.

Для этого:

- информация об импортируемой сборке заносится в таблицу метаданных AssemblyRef,
- информация о каждом импортируемом типе - в таблицу TypeRef,
- информация о каждом импортируемом методе и поле – в таблицу MemberRef.
- Кроме того, сборка может импортироваться динамически через рефлексию.
- Техника компонентного программирования представлена в файле file4.doc.

Библиотека MSDN (Microsoft Developer Network), содержащая информацию о программировании с использованием компонентов

При сборке приложения из стандартных протестированных компонентов

- сокращаются затраты времени на программирование и
- повышается надежность разрабатываемых приложений.

Техника работы с мастером Applications представлена в приложении 4.

Создание собственных компонентов и их **сборка из сложных компонентов, находящихся в классах .NET Framework** рассматривается в следующих разделах библиотеки MSDN**:**

№	Раздел библиотеки MSDN	Содержание
1	Примеры программирования компонентов	Список примеров создания и реализации компонентов.
2	Примеры кода для программирования компонентов	Примеры кода для типичных задач.
3	Устранение неполадок при разработке элементов управления и компонентов	Способы решения типичных проблем.
4	Создание компонентов	Описание архитектуры компонентов, их разработки, реализации и связанной с ними терминология.
5	COM-взаимодействие в Visual Basic и Visual C#	Ссылки на концептуальные сведения и подробные сведения о реализации взаимодействия между объектами COM и .NET.
6	Создание компонентов передачи сообщений	Введение в компоненты, взаимодействующие с системными ресурсами Microsoft Message Queuing.
7	Создание компонентов файловой системы и таймера	Введение в компоненты, реагирующие на изменения в файлах и папках, инициирующие события по расписанию, отслеживающие процессы Windows или взаимодействующие с ними.
8	Создание компонентов наблюдения за системой	Введение в компоненты, взаимодействующие со счетчиками производительности и журналами событий Windows.
9	Создание компонентов службы каталогов Active Directory	Введение в компоненты, которые взаимодействуют со службой каталогов Active Directory, позволяя выполнять администрирование папок из приложения.
10	Создание компонентов установки	Введение в компоненты установки, позволяющие выполнять пользовательские действия при развертывании приложения.
11	Пространства имен модели компонентов в Visual Studio	Содержит список основных пространств имен в .NET Framework, использующихся для создания элементов управления и компонентов, а также ссылки на разделы с основными концепциями и соответствующие разделы справки.
12	Пространства имен служб Framework в Visual Studio	Общие сведения об основных пространствах имен в .NET Framework, используемых для взаимодействия с различными ресурсами серверной части (например, очередями сообщений и службами каталогов), а также ссылки на другие справочные разделы и разделы с концептуальными сведениями.

Примеры разработки компонентов в MSDN помогут создавать компоненты и элементы управления для форм Windows Forms:

№	Название и адрес примера	Содержание примера
1	Разработка компонента с использованием Visual C#	Описание разработки простого компонента в Visual Basic, которая включает взаимодействие между клиентом и компонентом, время жизни объекта и циклические ссылки, отладку клиентов и компонентов и использование общих методов и методов экземпляра.
2	Разработка простого однопоточного компонента с использованием Visual C#	Демонстрируется разработка простого многопоточного компонента в C#, объясняется, как работают потоки и как координировать несколько потоков в компоненте.
3	Примеры программирования компонентов	Список примеров создания и реализации компонентов.
4	Устранение неполадок при разработке элементов управления и компонентов	Способы решения типичных проблем.
5	Программирование с использованием компонентов	Подробные сведения о программировании и ссылки на разделы о программировании с использованием компонентов (в том числе компонентов, взаимодействующих с системными ресурсами Microsoft Message Queuing, файловой системой и службой каталогов Active Directory).
6	Разработка компонентов	Описание процесса создания компонентов и специализированных элементов управления для форм Windows Forms.

Компонентное программирование может быть реализовано с использованием как внутренних, так и внешних компонент.

Внутренние компоненты в DOT NET представлены, например, в виде компонентов форм.

Внешние компоненты – это программы, разработанные вне среды DOT NET. Один из вариантов оформления таких программ (внешних компонент) – динамические библиотеки (dll), например, библиотеки интерфейса windows API.

Обычно внешние компоненты расширяют возможности библиотеки классов .NET Framework

- Библиотека классов .NET Framework — это библиотека классов, интерфейсов и типов значений, которая предоставляет доступ к функциональным возможностям системы и составляет основу для создания приложений, компонентов и элементов управления .NET Framework.
- Для поддержания широкого диапазона функциональных возможностей управляемая среда .NET Framework предоставляет разработчикам возможность совершенствования системы на основе двух основополагающих разработок:
  - Общей системы типов и
  - Общеязыковой спецификации.
- Общая система типов – это модель, определяющая правила, которым следует общеязыковая среда выполнения при работе (объявлении, использовании и управлении) с типами. Общая система типов устанавливает оболочку, которая делает возможной межъязыковую интеграцию, является сырьем, из которого создаются библиотеки классов.
- Общеязыковая спецификация (CLS) определяет ряд программно контролируемых правил, которые регулируют взаимодействие типов, созданных на разных языках программирования.
- Чтобы обеспечить возможность писать любую управляемую библиотеку, CLS должна быть достаточно богатой. Однако, если вы обеспечиваете множество способов решения одной и той же задачи, пользователи вашей библиотеки классов могут запутаться при выборе пути разработки и использования.
- На основе Общей системы типов и Общеязыковой спецификации могут быть разработаны расширяющие возможности DOT NET.

Например, такими такими дополнительными разработками являются:

- **NUnit** — открытая среда юнит-тестирования приложений для .NET. Она была портирована с языка Java (библиотека JUnit). Первые версии NUnit были написаны на J#, но затем весь код был переписан на C# с использованием таких новшеств .NET, как атрибуты.
- ZedGraph - это компонент для рисования графиков под .NET Framework Его адрес: http://jenyay.net/Programming/ZedGraph
- Система Knowledge .NET проф. В.О. Сафонова из НИИ математики и механики http://www.knowledge-net.ru/documents/2005-11-15_article.pdf - это пакет для интеграции методов инженерии знаний и инженерии программ, расширения языка C# средствами представления фреймовых и продукционных знанийhttp://www.old.zetms.ru/programs_group3.htm

http://www.old.zetms.ru/programs_group3.htm

- ZETLab-Studio - мощная среда разработки пользовательских приложений для задач измерения, диагностики и автоматизации, реализованная на C#.
- библиотека HTML Agility Pack, содержащая своё пространство имён:

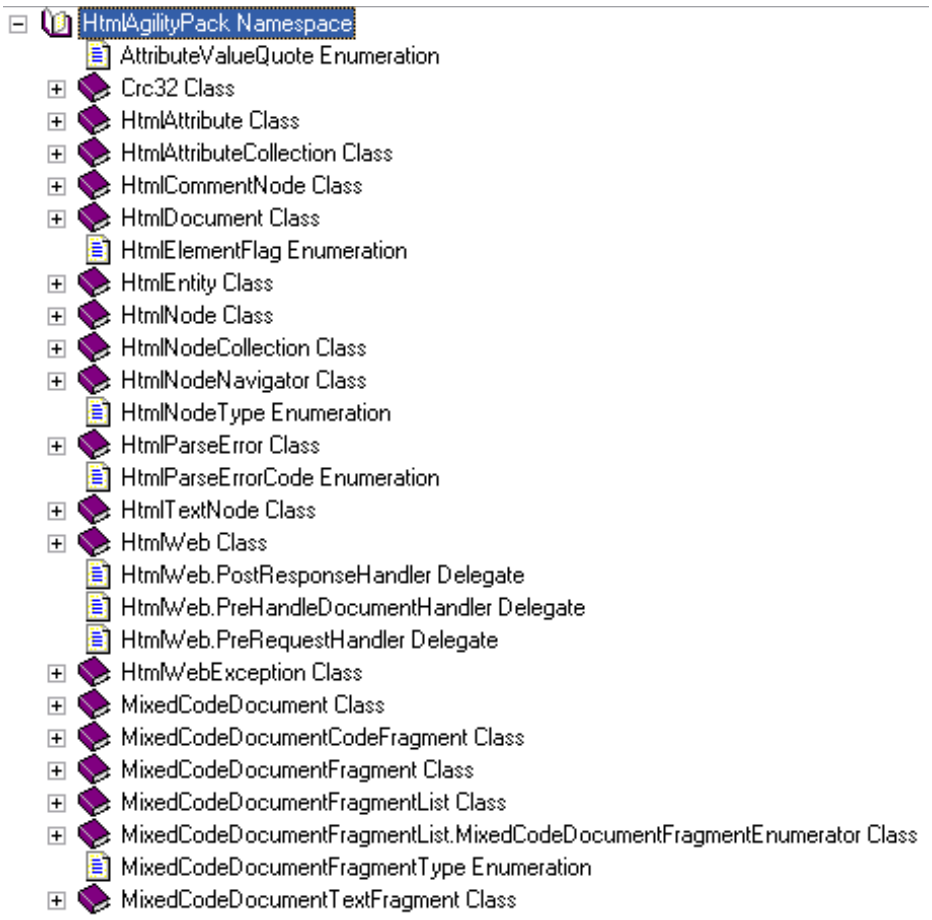


Рис. 28.

Библиотека предназначена для обработки HTML страниц.

- **Math.NET** представляет собой написанный на C# комплекс программ, предназначенный для символьных преобразований выражений в объектно-ориентированных средах.

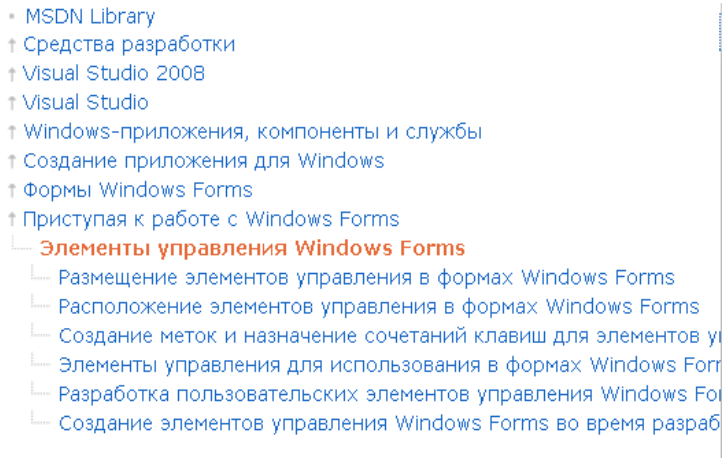
http://www.codeproject.com/KB/architecture/csdespat_1.aspx

- **GOF Design Patterns in C#** демонстрирует представление некоторых паттернов проектирования, использующих C#.

http://msdn2.microsoft.com/en-us/magazine/cc301786.aspx

- **Use P/Invoke to Develop a .NET Library for Serial Communications** Библиотека, использующая на C# платформу Invocation Services для прямого взаимодействия с Win32 API через последовательный интерфейс RS-232.
- По адресу: http://msdn.microsoft.com/ru-ru/vcsharp/Aa336740.aspx можно найти большое количество классов и библиотек, разработанных для использования совместно с dot Net, и в частности – с языком C#.

MSDN о компонентном программировании:



Примеры кода для программирования компонентов

В приведенном ниже списке представлены содержащиеся в файле категории примеров:

- Создание компонентов http://msdn.microsoft.com/library/rus/vbcon/html/vboricodeexamplesforcomponentprogramming.asp?frame=true - vboricodeexamplesforcomponentprogramminganchor2#vboricodeexamplesforcomponentprogramminganchor2
- Многопоточность http://msdn.microsoft.com/library/rus/vbcon/html/vboricodeexamplesforcomponentprogramming.asp?frame=true - vboricodeexamplesforcomponentprogramminganchor3#vboricodeexamplesforcomponentprogramminganchor3
- Контейнеры и узлы http://msdn.microsoft.com/library/rus/vbcon/html/vboricodeexamplesforcomponentprogramming.asp?frame=true - vboricodeexamplesforcomponentprogramminganchor4#vboricodeexamplesforcomponentprogramminganchor4
- Обработка исключений http://msdn.microsoft.com/library/rus/vbcon/html/vboricodeexamplesforcomponentprogramming.asp?frame=true - vboricodeexamplesforcomponentprogramminganchor5#vboricodeexamplesforcomponentprogramminganchor5
- Безопасность компонентов http://msdn.microsoft.com/library/rus/vbcon/html/vboricodeexamplesforcomponentprogramming.asp?frame=true - vboricodeexamplesforcomponentprogramminganchor6#vboricodeexamplesforcomponentprogramminganchor6
- Помощь пользователю http://msdn.microsoft.com/library/rus/vbcon/html/vboricodeexamplesforcomponentprogramming.asp?frame=true - vboricodeexamplesforcomponentprogramminganchor7#vboricodeexamplesforcomponentprogramminganchor7
- Элементы управления пользователя и специализированные элементы управления
- Компоненты сообщений http://msdn.microsoft.com/library/rus/vbcon/html/vboricodeexamplesforcomponentprogramming.asp?frame=true - vboricodeexamplesforcomponentprogramminganchor9#vboricodeexamplesforcomponentprogramminganchor9

- Компоненты файловой системы и таймера http://msdn.microsoft.com/library/rus/vbcon/html/vboricodeexamplesforcomponentprogramming.asp?frame=true - vboricodeexamplesforcomponentprogramminganchor10#vboricodeexamplesforcomponentprogramminganchor10
- Компоненты счетчика производительности http://msdn.microsoft.com/library/rus/vbcon/html/vboricodeexamplesforcomponentprogramming.asp?frame=true - vboricodeexamplesforcomponentprogramminganchor12#vboricodeexamplesforcomponentprogramminganchor12
- Компоненты журнала событий http://msdn.microsoft.com/library/rus/vbcon/html/vboricodeexamplesforcomponentprogramming.asp?frame=true - vboricodeexamplesforcomponentprogramminganchor13#vboricodeexamplesforcomponentprogramminganchor13
- Службы и процессы http://msdn.microsoft.com/library/rus/vbcon/html/vboricodeexamplesforcomponentprogramming.asp?frame=true - vboricodeexamplesforcomponentprogramminganchor14#vboricodeexamplesforcomponentprogramminganchor14
- Компоненты Active Directory http://msdn.microsoft.com/library/rus/vbcon/html/vboricodeexamplesforcomponentprogramming.asp?frame=true - vboricodeexamplesforcomponentprogramminganchor11#vboricodeexamplesforcomponentprogramminganchor11
- Примеры кодов

Резюме.

Определение термина компонент:

- Компонент предназначен для многократного применения.
- Компонент представляет собой отдельный модуль.
- Компонент определён, как независимый модуль.
- Компонент не является автономной программой.
- Функциями компонента может воспользоваться любое приложение.
- Программа, которая использует компонент, называется *клиентом* (client).
- Использовать компонент могут многие клиенты.
- Компонент может работать с любым количеством клиентов.
- Доступа к исходному коду компонента ни один клиент не имеет.
- Компонент должен быть представлен в двоичном формате.

Для того, чтобы клиент мог использовать компонент, необходимо, чтобы и клиент, и компонент подчинялись одному и тому же набору правил.

- Набор правил, определяющих форму и поведение компонента, называется **компонентной моделью** (component model).
- Компонентная модель в среде .NET Framework определяется интерфейсом **System.ComponentModel.IComponent**
- и классом **System.ComponentModel.Component**. Этот класс обеспечивает стандартную реализацию интерфейса IComponent по умолчанию.

Таким образом:

- При создании компонента используется класс System.ComponentModel.

- В компоненте создаётся своё пространство имён, которое потом для связки указывается в клиенте с помощью using.
- В компоненте создаётся класс – наследник класса Component из компонентной модели, в котором размещается содержание компонента. Экземпляр этого класса впоследствии создаётся и используется в клиенте.
- Связь клиента с компонентом осуществляется через пространство имён, созданное в компоненте и использование экземпляра класса, созданного в компоненте .
- Кроме того, связь клиента с компонентом указывается при компиляции клиента на уровне файлов, в которых расположены компонент и клиент.
- Компонент должен быть скомпилирован с получением dll, а не exe-файла.

Использование системы команд операционнй системы (ОС) из C#-программы

Классы .NET для работы с процессами

Пространство имён System.Diagnostics содержит 7 классов и 2 перечисления.

Для управления внешними (по отношению к Си-программам) программами основными являются 2 класса: Process и ProcessStartlnfo.

Класс Process предоставляет доступ к локальным и удалённым процессам и позволяет запускать и останавливать локальные системные процессы.

В MSDN каждый класс характеризуется по типовой схеме: общее описание класса, перечисление членов класса (конструктор, методы, свойства, события), подробное описание членов класса с примерами их использования. Такая же схема выдерживается и при описании классов Process и ProcessStartlnfo.

Электронной вычислительной машине для выполнения любой программы нужны определённые ресурсы: память, устройства, файлы, адресное пространство, процессорное время, и др. Программа, которой операционная система выделила необходимые для работы ресурсы, образует процесс, а точнее - процесс операционной системы, или системный процесс.

Последовательность создания процесса в C#-программе предусматривает создание экземпляра класса (компонента Process) и связывание этого экземпляра с системным процессом (т.е. процессом ОС).

Создание экземпляра производится конструктором класса. При этом экземпляру (компоненту, или созданному объекту) присваивается имя. Если процесс создаётся в классе MyProcess и ему присваивается имя process1, то делается это с использованием оператора new:

MyProcess process1 = new MyProcess();

Связывание создаваемого процесса с системным производится методом Start().

При создании процесса обязательно должно быть определено значение свойства Startlnfo.FileName. Обычно это свойство определяется перед связыванием с системным процессом (т. е. перед командой Start()) и определяет ресурс процесса - используемую программу или документ, обрабатываемый этой программой.

Например, создадим процесс из программы 100.cs, выводящей на экран help программы сетевого клиента ОС - cliconf.chm:

```
using System;
class Class1
{
[STAThread]
static void Main(string[]   args)
{
//  Создать  новый процесс
System.Diagnostics.Process proc = new System.Diagnostics.Process();
//  Приложение которое будем запускать
proc.StartInfo.FileName = "cliconf.chm";
proc.Start();
Console.ReadLine();
}
}
```

В этой программе конструктор System.Diagnostics.Process() создаёт экземпляр класса System.Diagnostics.Process с именем proc. Затем определяется требуемый этому процессу ресурс FileName = " cliconf.chm", и запускается метод Start().

Во время компиляции создаётся программа 100.exe, запуск которой приводит к появлению соответствующего системного процесса и появлению окна программы сетевого клиента:

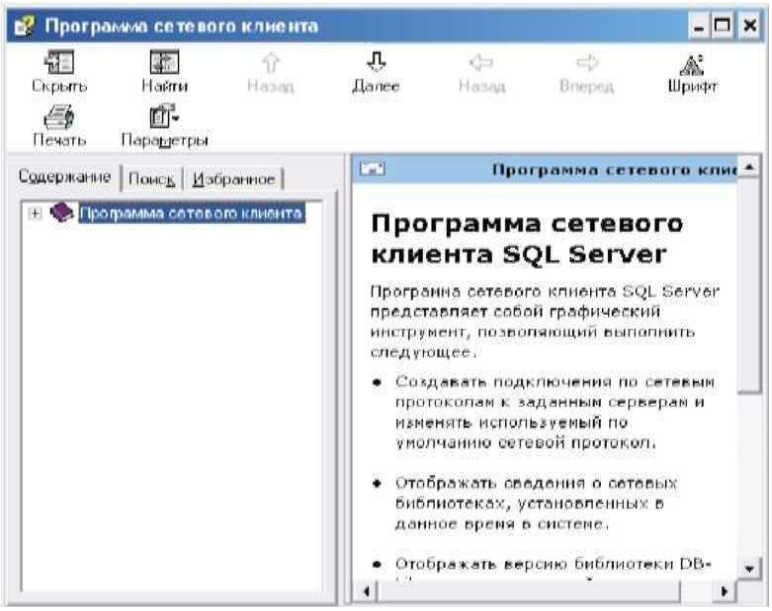


Рис. 29.

Системный процесс - это выполняющееся на компьютере приложение. Из C#-программы доступ к нему можно получить с помощью компонента Process (так в MSDN называется экземпляр класса Process, создаваемый с помощью конструктора).

Используя компонент Process, кроме доступа к системному процессу

- можно получить список выполняющихся на компьютере процессов или потоков,
- можно запустить новый процесс,
- остановить выполняющийся процесс,
- подключиться к другому процессу,
- получить информацию о нём: набор потоков, загруженные модули (файлы с расширениями dll и exe), информацию об объёме занимаемой памяти, временных параметрах выполнения процесса, и т.д.

Для запуска процессов в классе Process может использоваться 6 методов Start:

Start()	Запускает (или начинает заново использовать) ресурс процесса, определенный свойством StartInfo этого компонента Process, и связывает его с компонентом.
Start(ProcessStartInfo)	Запускает ресурс процесса, определенный параметром, содержащим стартовую информацию процесса (например, имя файла процесса для запуска), и связывает ресурс с новым компонентом Process.
Start(String)	Запускает ресурс процесса путем указания имени документа или файла приложения и связывает ресурс с новым компонентом Process.
Start(String, String)	Запускает ресурс процесса путем указания имени приложения и множества аргументов командной строки и связывает ресурс с новым компонентом Process.
Start(String, String, SecureString, String)	Запускает ресурс процесса путем указания имени приложения, имени пользователя, пароля и домена и связывает ресурс с новым компонентом Process.
Start(String, String, String, SecureString, String)	Запускает ресурс процесса путем указания имени приложения, набора аргументов командной строки , имени пользователя, пароля и домена и связывает ресурс с новым компонентом Process.

Среди этих методов только метод Start() не является статическим.

Перед его использованием необходимо создать экземпляр класса Process, в свойстве StartInfo надо задать информацию для определения запускаемого ресурса процесса.

Причём, в этом свойстве можно задать всё, что можно передать программе в командной строке (обязательным является только значение свойства FileName).

Наиболее важные классы, свойства и методы, используемые в пространстве имён System.Diagnostics:

1. Свойство FileName http://www.novainfo.ru/kompleksirovanie-programmnykh-sredstv/4/1/2
2. Класс ProcessStartInfohttp://www.novainfo.ru/kompleksirovanie-programmnykh-sredstv/4/1/3
3. Класс Process http://www.novainfo.ru/kompleksirovanie-programmnykh-sredstv/4/1/4
4. Свойство StartInfo http://www.novainfo.ru/kompleksirovanie-programmnykh-sredstv/4/1/5
5. Свойство RedirectStandardError http://www.novainfo.ru/pravovaya-kvalifikatsiya-derivativov/4/1/6
6. Свойство RedirectStandardInput http://www.novainfo.ru/pravovaya-kvalifikatsiya-derivativov/4/1/7
7. Свойство RedirectStandardOutput http://www.novainfo.ru/pravovaya-kvalifikatsiya-derivativov/4/1/8
8. Свойство StandardOutputEncoding http://www.novainfo.ru/pravovaya-kvalifikatsiya-derivativov/4/1/9
9. Свойство ProcessStartInfo.UseShellExecut http://www.novainfo.ru/pravovaya-kvalifikatsiya-derivativov/4/1/10
10. Свойство ProcessStartInfo.Verb http://www.novainfo.ru/pravovaya-kvalifikatsiya-derivativov/4/1/11
11. Свойство ProcessStartInfo.Verbs http://www.novainfo.ru/pravovaya-kvalifikatsiya-derivativov/4/1/12
12. Свойство ProcessStartInfo.WorkingDirectory

1. Свойство FileName

Может определять либо исполнимый файл, либо ресурс для него (т.е. обрабатываемый документ).

Что именно должно быть задано этим свойством, уточняется свойством UseShellExecute класса ProcessStartlnfo.

2. Класс ProcessStartlnfo

Используется совместно с классом Process.

Эти два класса так близки, что в MSDN они почти не различаются.

3. Класс Process

Имеет свойства, характеризующие заголовок системного процесса, его идентификатор (PID), приоритет, объём занимаемой физической памяти, состав используемых модулей, время начала и окончания процесса, и другие параметры.

4. Свойство StartInfo

Позволяет передавать параметры запускаемому процессу. Основным является параметр FileInfo. Это может быть исполняемая программа ОС или ресурс.

В составе Startlnfo в свойстве Verb можно указать команду, которую должна выполнить над ресурсом операционная система. Например, при обработке файла, имеющего расширение .doc, можно выполнить команду "Print", а при обработке файла с расширением .exe можно выполнить команду runas.

Состав команд можно определить свойством Verbs.

Параметры, указываемые в свойстве StartInfo, можно изменить только до момента вызова метода Start. После запуска процесса изменение значений StartInfo не влияет на запущенный процесс.

Для управления машинными программами представляют интерес такие свойства, как Standardlnput, StandardOutput, StandardError (позволяют взаимодействовать со стандартными потоками ввода, вывода и ошибок).

5. Свойство RedirectStandardError

Позволяет перенаправить поток ошибок, возникающих при выполнении процесса.

Для изменения направленности потока ошибок свойству RedirectStandardError надо присвоить значение true, и одновременно необходимо запретить использование средств операционной системы, присвоив свойству UseShellExecute значение false.

6. Свойство RedirectStandardInput

Позволяет перенаправить входной поток.

Стандартным устройством ввода является клавиатура. Его можно заменить на файл - при этом свойству RedirectStandardInput присваивается значение true.

В следующем примере программой 116.cs перенаправляется поток процесса Standardlnput - запускается команда sort с перенаправленным вводом. (Команда sort является консольным приложением, считывающим и сортирующим введённый текст).

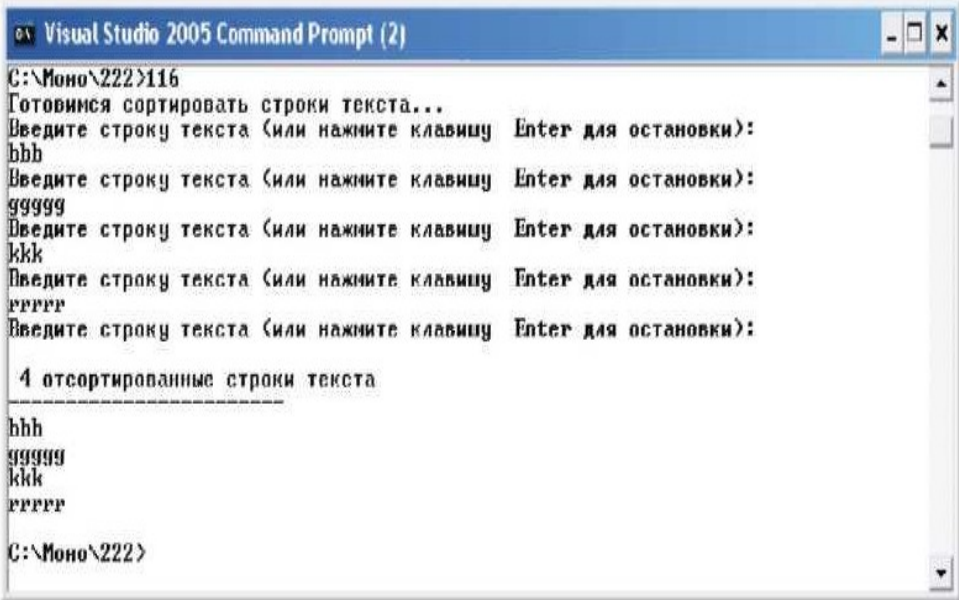


Рис. 30.

Если эту программу исполнить без перенаправления, т.е. закомментировать две строки:

myProcess.StartInfo.FileName = "Sort.exe"; //myProcess.StartInfo.UseShellExecute = false; //myProcess.StartInfo.Redirect StandardInput = true;

myProcess.Start();

то программа 116.exe будет выполняться в окне Visual Studio Command Prompt:

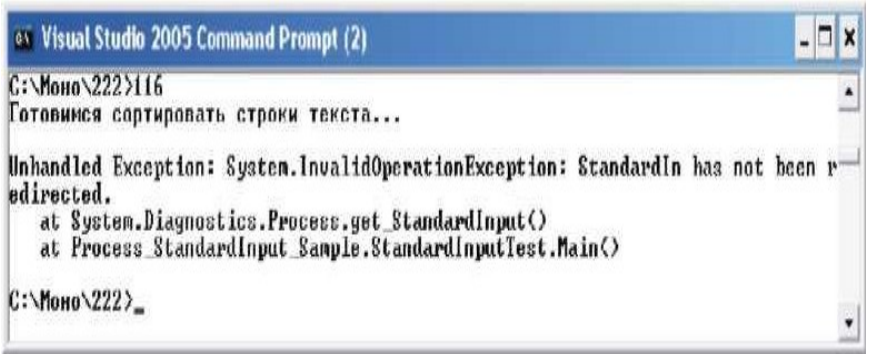


Рис. 31.

а строки для ввода будут восприниматься из окна запущенной программы sort.exe.

7. Свойство RedirectStandardOutput

Это свойство позволяет перенаправить выходной поток.

Обычно выводимый в выходной поток текст поступает в консоль (выводится на экран).

Перенаправление потока эквивалентно командам ">", "»" или "|" операционной системы.

ProcessStandardOutput - это стандартный поток компонента Process.

Свойство RedirectStandardOutput - это перенаправление стандартного потока.

Если поток надо перенаправить, то это свойство должно иметь значение true.

Если перенаправлять ничего не надо, свойство должно быть равно false.

Совместно с RedirectStandardOutput устанавливается свойство UseShellExecut.

При RedirectStandardOutput = true свойство UseShellExecut должно быть false.

8. Свойство StandardOutputEncoding

- Возвращает или задает предпочтительную кодировку для стандартного вывода.
- Если значение этого свойства равно null, процесс использует для стандартного вывода соответствующую кодировку по умолчанию.

9. Свойство ProcessStartInfo.UseShellExecut

- Возвращает или задает значение, позволяющее определить, нужно ли использовать оболочку операционной системы для запуска процесса.
- Для использования оболочки системы при запуске процесса, свойство должно иметь значение true. В противном случае процесс создаётся непосредственно из исполняемого файла, а возможности ОС не используются.

При использовании оболочки ОС запускать процесс можно по имени обрабатываемого документа.

По этому имени оболочка ОС определяет тип документа и находит в реестре, какая программа является обработчиком документов данного типа, после чего -запускает эту программу и передаёт ей в качестве параметра обрабатываемый файл.

Кроме того, при использовании оболочки ОС файл может в некоторых случаях задаваться без указания пути к нему. Если путь не указан, файл сначала ищется в текущем каталоге, а при отрицательном исходе - в каталогах, перечисленных в системной переменной path.

К особенностям использования оболочки ОС относится и способность ОС определять приоритет запуска файлов, имеющих одинаковые имена, но разные расширения.

Например, в текущем каталоге хранится три файла: "a.cmd", "a.com" и "a.exe". Если процесс запускается без указания расширения, только по имени файла, при UseShellExecute = true операционная система сначала выполнит файл с расширением ".com", затем - ".exe", и только затем - ".cmd".

При UseShellExecute = false файлы будут выполняться в том порядке, в котором они были записаны в каталог.

Задав для этого свойства значение false, можно перенаправлять потоки ввода, вывода и ошибок.

К особенностям использования оболочки ОС относится и способность ОС определять приоритет запуска файлов, имеющих одинаковые имена, но разные расширения.

Например, в текущем каталоге хранится три файла: "a.cmd", "a.com" и "a.exe". Если процесс запускается без указания расширения, только по имени файла, при UseShellExecute = true операционная система сначала выполнит файл с расширением ".com", затем - ".exe", и только затем - ".cmd".

При UseShellExecute = false файлы будут выполняться в том порядке, в котором они были записаны в каталог.

Задав для этого свойства значение false, можно перенаправлять потоки ввода, вывода и ошибок.

2.1. Запуск командного процессора

Для запуска 32-разрядного командного процессора операционной системы создаём процесс с идентификатором my, в свойстве FileName указываем полный адрес командного процессора и запускаем процесс:

В результате появляется окно командного процессора.

```
using System;
using System.Diagnostics;
class MyProcess
{
public static void Main()
{
Process my = new Process();
my.StartInfo.FileName = "C:\\WINDOWS.0\\system32\\cmd.exe"
my.Start();
Console.Read(); }
}
```

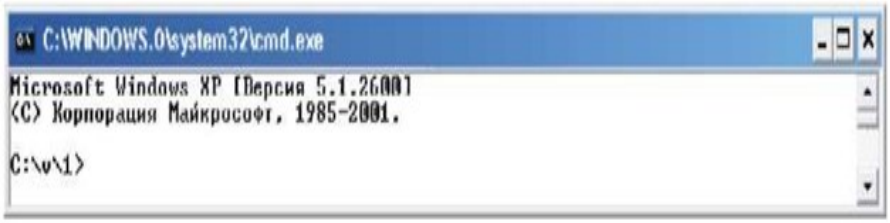


Рис. 32.

2.2. Ресурсы процесса и передача параметров запускаемому процессу

В операционной системе Windows запустить процесс можно двумя способами:

- Запустить обрабатывающую программу и с помощью параметров передать этой программе имя обрабатываемого документа;
- Дважды щёлкнуть по файлу, содержащему обрабатываемый документ.

Операционная система по расширению файла определит, какая программа обрабатывает документ такого типа, вызовет на исполнение эту программу и передаст ей обрабатываемый документ.

Вызов программы: указанием ресурса.

C#-программа processstartinfo2.exe используется для запуска файла, содержащего исходный код программы.

Запуск процесса производится указанием имени запускаемого ресурса.

Поскольку имя программы для обработки данного ресурса не указывается, необходимо разрешить операционной системе использовать свои возможности.

Для этого свойство StartInfo.UseShellExecute получает значение true.

Все необходимые файлы находятся в папке v.

Протокол запуска программы:

C:\v>processstartinfo2

C: \v>

На экране появляется окно Word, содержащее исходный код программы processstartinfo2.es.

```
using System;
using System.Diagnostics;
class MyProcess
{
public static void Main()
{
Process myProcess = new Process();
try
{
myProcess.StartInfo.UseShellExecute = true;
myProcess.StartInfo.FileName = "C:\\v\\processstartinfo2.doc";
myProcess.Start() ;
}
catch   (Exception e) {
Console.WriteLine(e.Message) ;
}
}
}
```

2.3. Перехват результатов работы процесса

Процесс создаётся для того, чтобы выполнить какую-то часть работы, а затем использовать полученный результат.

С этой целью выполним программу 7.cs:

```
//Программа 7.cs
using System;
using System.Diagnostics;
class MainClass {
public static void Main() {
Process p = new Process();
p.StartInfo.FileName = "cmd.exe";
p.StartInfo.Arguments = "/c dir *.cs";
p.StartInfo.UseShellExecute = false;
p.StartInfo.RedirectStandardOutput = true;
p.Start();
string output = p.StandardOutput.ReadToEnd();
Console.WriteLine("Output:");
Console.WriteLine(output); }
}
```

- В этой программе с помощью 32-разрядного процессора cmd.exe запрашивается, какие исходные тексты программ, написанных на языке c# содержатся в текущем директории. Имеется в виду директорий v: в корне диска C:.
- В свойстве StartInfo.Arguments содержится комбинация символов /c, обозначающая необходимость выполнения команды операционной системы - далее указывается, что такой командой должна быть команда dir. Результат выполнения этой команды разрешено перенаправить в стандартный выходной поток.
- Последние три команды посвящены работе с этим потоком.
  - Сначала стандартный выходной поток направляется в переменную output.
  - Затем выводится на печать заголовок, предупреждающий, что будет выводиться содержимое стандартного выходного потока, т. е. результат работы команды dir операционной системы.
  - И затем распечатывается содержание переменной output, т. е. список имеющихся в каталоге у исходных текстов сопрограмм.

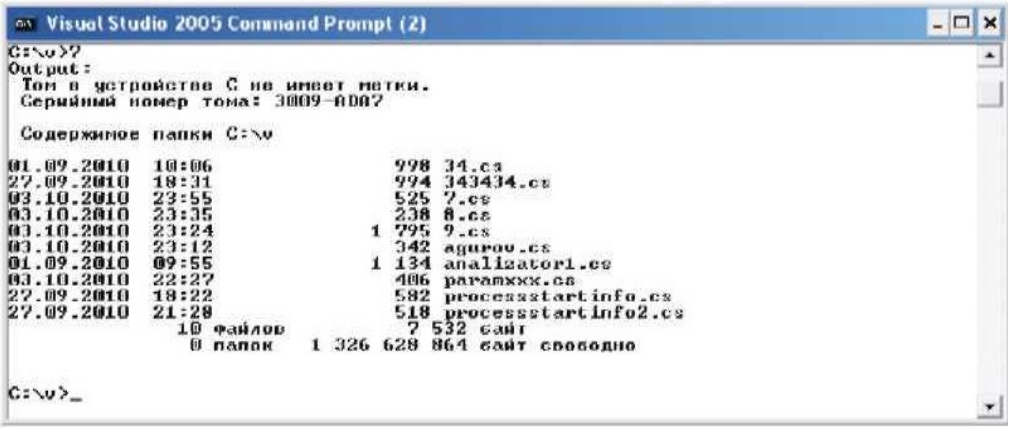


Рис. 33.

Запуск команд ОС из C#-программы

1. Установка атрибутов файла http://www.novainfo.ru/kompleksirovanie-programmnykh-sredstv/4/3/2
2. Получение структуры папки по команде tree http://www.novainfo.ru/kompleksirovanie-programmnykh-sredstv/4/3/3
3. Вывод на экран текстового файла командой type http://www.novainfo.ru/kompleksirovanie-programmnykh-sredstv/4/3/4
4. Создание виртуального диска http://www.novainfo.ru/kompleksirovanie-programmnykh-sredstv/4/3/5
5. Управление работой архиватора http://www.novainfo.ru/kompleksirovanie-programmnykh-sredstv/4/3/6
6. Создание командного файла из C#-программы. http://www.novainfo.ru/kompleksirovanie-programmnykh-sredstv/4/3/7

7. Вывод на экран созданного файла

Комплексирование программных средств с помощью командного файла ОС (command, CMD-файла)

При комплексировании программных средств (ПС) одна из основных проблем – обмен данными между совершенно несвязанными между собой программами, исходные тексты которых – неизвестны.

Если известно, на каком языке были написаны исходные тексты, и каким компилятором были получены .exe-модули, то можно попробовать дизассемблирование для восстановления исходных текстов, и затем – внесение необходимых изменений и дополнений в исходные тексты.

Но чаще всё это недоступно.

Постановка задачи:

Известно, что программа «А» выдаёт несколько необходимых нам данных, и группу ненужных.

Программа «В» выдаёт несколько других необходимых нам данных, и тоже группу ненужных.

Задача заключается в том, что
и в результатах, полученных программой «А»,
и в результатах, полученных программой «В»
надо выделить необходимые нам данные, и соединив их, представить программе «С».

Другими словами, нам необходимо:

1. Перехватить результаты работы каждой программы и сохранить их в файле
2. Или перехватывать и перенаправлять выходные потоки программ.
3. Выделять из перехваченных данных необходимые для дальнейшего использования
4. Соединять данные, полученные из разных источников
5. Подготовленные таким образом данные передавать на вход программы для их дальнейшей обработки.

Все эти задачи должны выполняться автоматически, без перенабора данных на клавиатуре.

Практически любая программа работает по следующей схеме:

«Перехватить поток» - это значит перенаправить его туда, где он необходим.

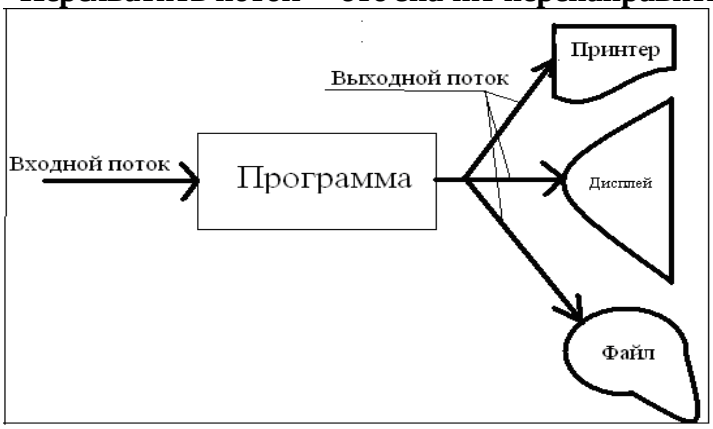


Рис. 34.

Перенаправление потока в операционной системе Windows выполняется с помощью трёх команд:

<
>
>>

Команда “<” - это команда перенаправления **входного** потока.
Острие команды показывает, куда она направлена.
Например: команда

- debug.exe < file.txt

направляет входной поток из файла file.txt на вход программы debug.exe.
Это значит, что в файле file.txt записана последовательность команд, которые должна выполнить программа debug.exe.
C:\4>debug < 1.txt > 2.doc

Рис. 35.

Команда “>” это команда перенаправления выходного потока.
Острие команды так же показывает, куда она направлена.
Например:

- Debug.exe > file1.txt.

означает, что результаты работы программы debug.exe будут направляться не на экран (как обычно), а в файл file1.txt.
Если этого файла не существует, он будет создан. А если он существует, его содержимое будет заменено результатами работы программы debug.exe.
Т.е. информация в этот файл поступит, начиная с самого его начала.
Команда “>>” это команда перенаправления выходного потока.
Острие команды так же показывает, куда она направлена.
Например:

- Debug.exe >> file2.txt.

означает, что результаты работы программы debug.exe будут направляться не на экран, а в файл file2.txt.
Если этого файла не существует, он будет создан.
А если он существует, его содержимое будет дополнено результатами работы программы debug.exe.
Причём эти результаты ничего не уничтожат в файле, они дополнят старое содержимое файла (новые данные будут дописаны в его конец).

Конвейер команд

Выделять из перехваченных данных необходимые для дальнейшего использования можно, используя конвейер команд.

Рассмотрим выдачу, полученную командой dir для корневого директория диска C:\.

28.03.2006	23:31	<DIR>	777
25.01.2007	19:16	<DIR>	Из HSE
17.09.2005	15:56	<DIR>	2
10.09.2005	16:05	<DIR>	1
21.11.2005	22:47	<DIR>	PerfLogs
04.12.2005	19:24	<DIR>	temp
15.01.2006	16:57	<DIR>	2006н
02.03.2006	18:12	<DIR>	Magic Waterfall Screensaver
04.03.2006	13:52	<DIR>	Test_System
02.05.2006	16:22	<DIR>	WebServers
02.02.2004	22:26	<DIR>	DVD

```
17.05.2006  15:54          410 543 braun_dyen_kod_da_vinchi.rtf.zip
11.05.2006  16:19          184 320 Olympus 310.doc
06.10.2006  11:12           26 624 hakers word.doc
03.01.2005  16:44    <DIR>         1c
25.03.2004  11:09    <DIR>         ACTIVstudio
21.02.2005  18:30              182 my_first.pl
18.07.2005  13:07    <DIR>         2005-2006 уч год
11.08.2005  11:48    <DIR>         JPG-BMP
               9 файлов        622 331 байт
              29 папок     254 394 368 байт свободно

C:\>
```

Рис. 36.

Надо отметить, что этот директорий очень напряжённый и насыщенный. Чаще всего он по команде dir не умещается в окне консоли команд.
Здесь приведен лишь фрагмент вывода по команде dir. Обратим внимание на выделенное красным цветом.
Получается, что кроме огромного перечня файлов и папок команда dir даёт и сводные результаты:

- в директории хранится 9 файлов
- общим объёмом 622 331 байт, и
- 29 папок.

При этом на диске, в корневой директорий которого мы заглядывали,

- свободно 255 000 576 байт.

Это значит, что команда dir даёт достаточное количество информации, чтобы ответить на такие вопросы:

- Сколько файлов содержится в данном каталоге?
- Какой объём занимают файлы, хранящиеся в данном каталоге?
- Сколько папок содержится в данном каталоге?
- Сколько свободного места есть на данном диске?

Но чтобы получить ответы на эти вопросы, надо переворошить груду информации и найти в ней то, что Вас интересует. Ответ должен быть коротким, понятным и информативным.
Чтобы удовлетворить этим требованиям, одной команды dir недостаточно.

Возможности этой (и других команд) можно расширить с помощью конвейера.

Вспомним, что есть ещё такая команда find, по которой можно среди большого количества строк найти ту, которая больше всего интересует.
Конвейер команд позволяет выдачу, полученную по одной команде, предоставить для анализа другой команде.
Построим конвейер из двух команд: dir и find.
Первая команда получит информацию о корневом директории диска C:, а вторая позволит выделить из этой информации (иногда – очень большого объёма) нужную.
Такое сочетание команд в конвейере позволяет ответить на вопрос:
«Сколько файлов хранится в данной директории и каков их объём?».
Выполним в консоли команд:

- C:\>dir | find "файлов"
  19 файлов 136 257 байт

Здесь для команды find в кавычках мы указали поисковое предписание: что искать?
А если вместо слова «файлов» указать слово «байт»?

- C:\>dir | find "байт"
  19 файлов 136 257 байт

Получили то же самое.

Внутри одной строки программа find неспособна разделить информацию?
Попробуем ответить на другой вопрос: «Сколько папок содержится в данном каталоге?».
Для этого команде find нужно указать другое поисковое предписание: «папок».

- C:\>dir | find "папок"
  29 папок 254 394 368 байт свободно

А что, если в качестве поискового предписания задать расширение файлов? Например, «.doc»

- C:\>dir | find ".doc"
- 11.05.2006 16:19 184 320 Olympus 310.doc
- 06.10.2006 11:12 26 624 hakers word.doc
- C:\>

На такое поисковое предписание машина вывела всю известную ей информацию о файлах с расширением «.doc» в корневом каталоге диска C:.
Но её опять слишком много – здесь и дата 11.05.2006 и 06.10.2006 , и время 16:19 и 11:12, и размеры файлов (184 320 и 26 624), и полные названия.
А что представляет собой программа find?
Проведём её исследование.
В основу исследования положим подсказку Windows, полученную
из help:

- find /?

или

- help find

Обратим внимание на атрибут /C. Применим его вместе с командой find:

- C:\>dir | find /C ".doc"
  2
- C:\>

В отличие от предыдущего обращения здесь конкретно указывается, сколько файлов с расширением .doc содержится в искомой папке.

Программа findstring.

Аналогичный результат может быть получен и по другой команде:
findstr.

- C:\>dir AX1 | findstr "файлов"
  25 файлов 3 444 924 байт

Что ещё можно получить от команды dir?
Листинг этой команды содержит ещё и такие данные, как «дата» и «время» для каждого файла и папки.
Поэтому в принципе можно получить конкретный ответ на вопрос:
Когда был записан файл …
или Когда была сформирована папка …

Команда ipconfig

Команда ipconfig без параметров сообщает IP-адрес Вашего компьютера.
До подключения к Интернет она покажет:

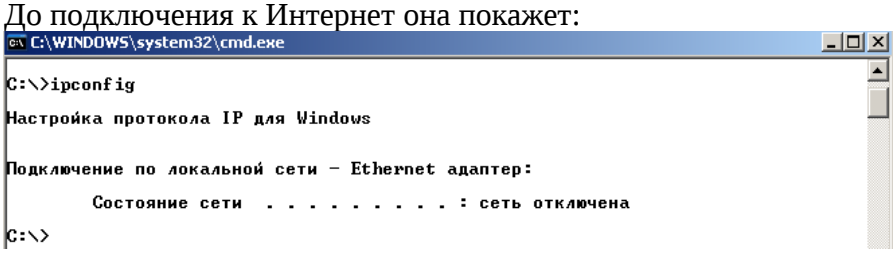


Рис. 37.

После подключения:

```
C:\WINNT\system32\cmd.exe

C:\>ipconfig

Настройка протокола IP для Windows 2000

Адаптер Ethernet Подключение по локальной сети:

        DNS суффикс этого подключения . . :
        IP-адрес  . . . . . . . . . . . . : 192.168.1.10
        Маска подсети . . . . . . . . . . : 255.255.255.0
        Основной шлюз . . . . . . . . . . :
```

Рис. 38.

Применим конвейер для выделения только IP-адреса:

```
C:\WINNT\system32\cmd.exe

C:\>ipconfig | find "IP-адрес"
        IP-адрес  . . . . . . . . . . . . : 192.168.1.10
```

Рис. 39.

Сохраним полученный IP-адрес в текстовом файле:

```
C:\WINNT\system32\cmd.exe

C:\>ipconfig | find "IP-адрес" > 123456.txt

C:\>type 123456.txt
        IP-адрес  . . . . . . . . . . . . : 192.168.1.10
```

Рис. 40.

Или если прочитать в блокноте:

```
123456.txt - Блокнот
Файл  Правка  Формат  Справка
        IP-адрес. . . . . . . . . . . . : 192.168.1.10
```

Рис. 41.

Таким образом, для выделения только необходимой информации можно воспользоваться конвейером команд и стандартными программами операционной системы Windows.

Если возможности стандартных программ не устраивают, небольшие программки для реализации требуемых операций можно написать и на C#.

Главное, что **этот метод позволяет комплексировать любые программы, а не только те, на которые ориентирована технология .NET**.

Технология XP (eXtream Programming) Кента Бека.

Историческая справка

Согласно утверждению Кента Бека (Kent Beck), Уорда Каннингема (Ward Cunningham) и Рона Джефриса (Ron Jeffries), авторов методики экстремального программирования,

эта методика представляет собой не столько следование каким-то общим схемам действий, сколько применение комбинации специальных техник.

При этом каждая техника важна, и без ее использования разработка считается идущей не по XP

Экстремальное программирование - это дисциплина разработки программных средств и ведения бизнеса в области создания программных продуктов,

которая фокусирует усилия программистов-разработчиков и бизнесменов-заказчиков на общих и вполне достижимых целях.

Бизнесменам как можно быстрее нужно работающее программное обеспечение в качестве конкурентного преимущества в бизнесе.

Программисты хотят заработать денег и самореализоваться , удовлетворив требования заказчика к качеству программного продукта.

Естественное желание первых как можно быстрее получить конкурентное преимущество,

согласуется с естественным желанием программистов упростить технологический процесс разработки.

Экстремальное программирование стоит применять, если имеют место:

- Неясные или изменяющиеся требования к системе
- Ответственные и квалифицированные разработчики
- Понимающий заказчик, который согласен участвовать в разработке.

От экстремального программирования лучше отказаться, если:

- Команда разработчиков более 50 человек
- Контракт на разработку строго регламентирует объем работ и, как следствие, имеет фиксированную стоимость.
- Разрабатываемое программное средство должно быть высоконадежным в плане обеспечения безопасности применения (относится к категории критических программных систем).

Модель экстремальной разработки программных средств

Технология экстремальной разработки использует четыре базовых технологических процесса с очень короткими повторениями в цикле (рис.):

- выслушивание заказчика,
- планирование,
- кодирование,
- тестирование.

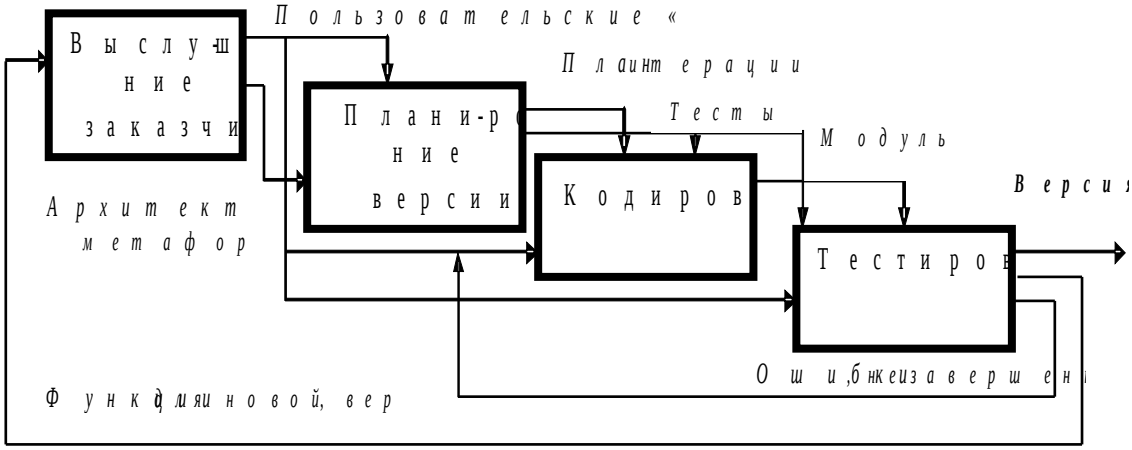


Рис. 42.

Выслушивание заказчика практически снимает проблему традиционной спецификации требований.

Сформулированные заказчиком (пользователем) исходные требования к программной системе фиксируют в виде так называемых пользовательских «историй».

Пользовательская «история» - это два-три предложения, которые в письменном виде излагают заказчики-пользователи и содержат действия, которые должна выполнять для них программная система.

Большей детализации требований до момента начала кодирования не требуется

Если программная система сложная, и предусматривает выполнение большого количества функций, пользовательские истории дробят весь комплекс функций на версии.

В каждую версию включаются самые важные, первоочередные функции (истории), без которых в данный момент нельзя обойтись.

Пользовательские истории используют для разработки программ (кодирования) и тестов (тестирования).

С началом реализации «истории» могут уточняться.

Реализация не представляет собой какой-то монолитный процесс – она делится на версии.

Выпуск версии - это очередной этап реализации разработки, в процессе которого создаётся функционально законченная часть программного средства.

При выпуске версии могут быть выявлены новые функции программной системы, которые еще не отражены в пользовательских «историях» (т.е. изменена функциональность).

Реализация разработки ведётся итерациями.

Для выявляемых новых функций составляются новые пользовательские истории.

Заказчик выбирает «истории» для очередной итерации с учетом важности описываемых функций и их логической связи.

Каждая итерация продолжается от одной до трех недель.

Для первой версии выбирают небольшое количество логически связанных и наиболее важных пользовательских «историй».

Для каждой последующей версии разумно выбирать наиболее важные и связанные между собой «истории» из числа оставшихся нереализованными.

Таким образом, цель каждой версии – подключение к реализации новой порции «историй».

Теоретически продолжительность проекта определяется тем, какое количество пользовательских историй нужно реализовать, и с какой скоростью они могут быть реализованы.

Скорость проекта - это количество пользовательских историй фиксированного объема, которые могут быть реализованы за одну итерацию.

Скорость проекта является основной метрикой экстремального программирования

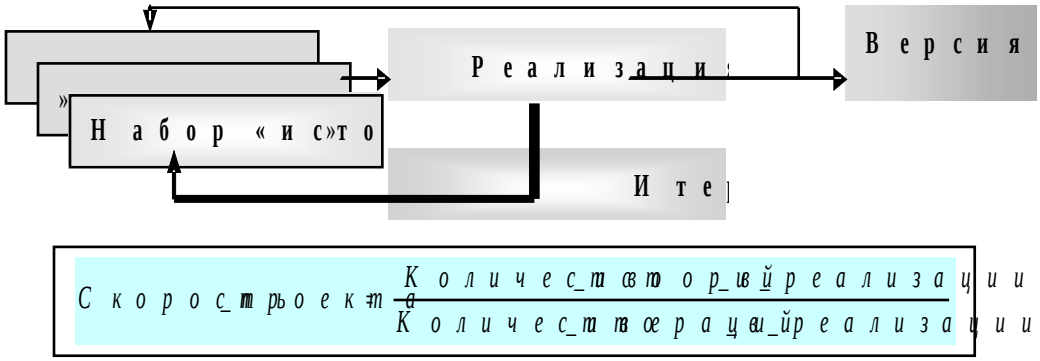


Рис. 43.

Практики (методы) экстремального программирования

К числу общеизвестных относят 12 практик (методических приема) экстремального программирования:

1. Игра планирования - быстрое определение области действия итераций разработки.

2. Частая смена версий с быстрым вводом в эксплуатацию упрощенной версии и малым периодом итерации разработки последующих версий.
3. Использование метафорического образа, или архитектурной метафоры - простого и доступного описания легенды работы и структуры программной системы, основанного на метафоре аналогий с известными процессами и структурами.
4. Максимально упрощенное в контексте кодирования проектирование.
5. Опережающая разработка тестов самими разработчиками по «пользовательским» историям до начала кодирования.
6. Рефакторинг (реорганизация) программного кода с целью его внутреннего улучшения (неизбыточность, гибкость, простота), но без изменения внешнего поведения программы.
7. Парное (дуальное), защищающее от ошибок программирование, когда программный код пишется одновременно двумя разными программистами;
8. Коллективное владение кодом, когда любой разработчик может улучшать программный код системы в любое время.
9. Непрерывное сборочное тестирование, когда система тестируется после каждого добавления нового модуля;
10. Сорокачасовая рабочая неделя без сверхурочных работ.
11. Постоянное присутствие заказчика в группе разработчиков.
12. Применение стандартов программирования.

В методическом плане дисциплина экстремального программирование допускает применение представительных подмножеств (например, 1,2,3,4,5,7,8,9) перечисленных выше практик.

Использование даже не всех, а только некоторых основных из этих практик можно расценивать как экстремальное программирование или, по крайней мере, как полезное явление в плане обеспечения адаптивности разработки.

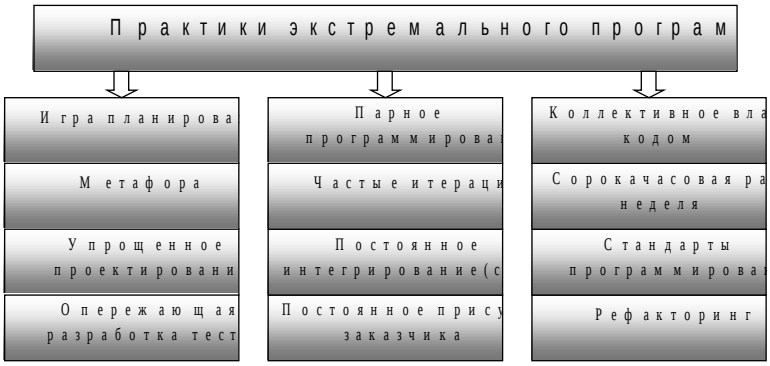


Рис. 44.

Рассматривая в качестве классификационных признаков виды деятельности, практики экстремального программирования делят на три группы (Уильям Вейк):

– Организационные
– Командные
– Индивидуальные

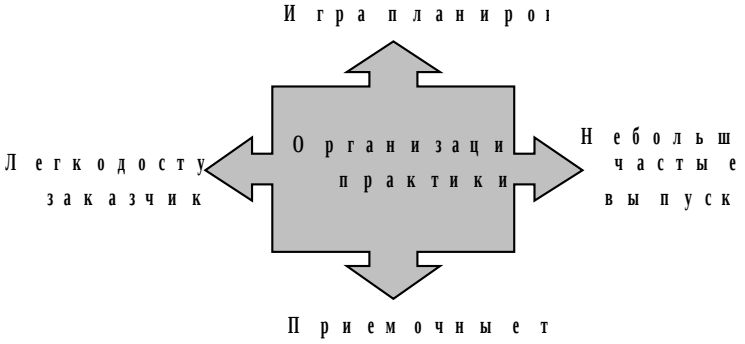


Рис. 45.

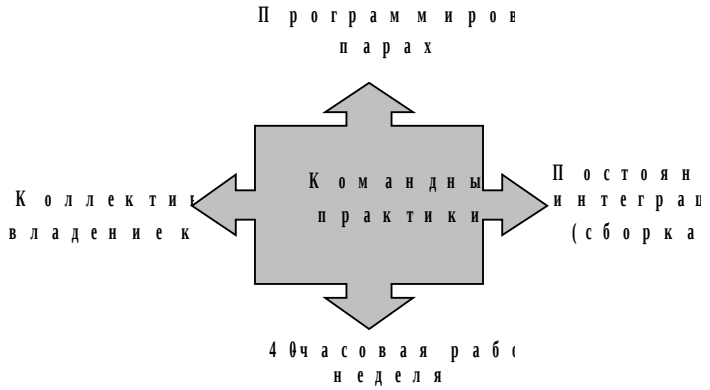


Рис. 46.

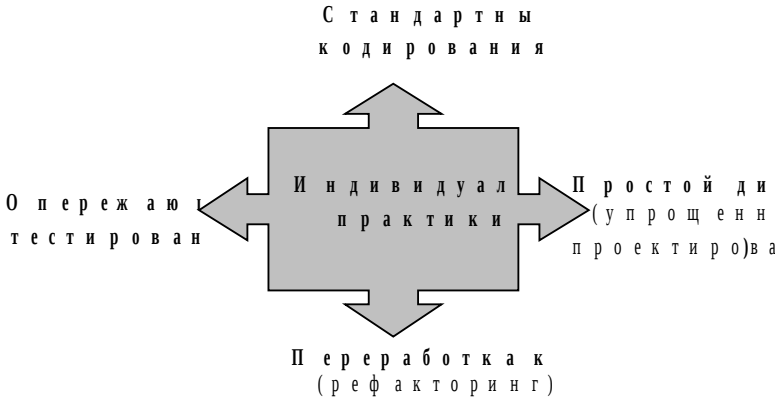


Рис. 47.

В каждой группе по четыре практики-текущих вида деятельности. Они отражают так называемый «холонический» подход к экстремальному программированию.
Термин «холонический» образован из двух слов - the whole (целое) и on (часть).
Поэтому «Холонический» подход характеризуется как организация работы, способствующая сохранению индивидуальности при функционировании как части единого целого.
Отсюда следует, что все четыре группы используют индивидуально, но в рамках единой разработки.

Организационные практики влияют на всю организацию в целом. Поэтому любые изменения должны быть согласованы на организационном уровне.

Командные практики влияют только на деятельность команды, участвующей в разработке.
Отдельные члены команды могут применять свой подход для решения отдельных задач.

Индивидуальные практики имеют самые большие возможности внедрения XP в личную практику разработчиков - применение стандартов, переработку, проектирование, тестирование.

Деятельность команды заказчиков

Главная обязанность заказчика заключается в принятии бизнес-решений.

Принятие бизнес-решений имеет отличие от традиционного определения требований к программе по своему содержанию.

Бизнес решения - это и определение требований, и оценка выгоды от реализации требований, и оценка стоимости их реализации.

При традиционном определении требований их состав фиксируют до начала проектирования, а стоимость разработки заранее определена договором (контрактом).

Так как экстремальное программирование представитель гибкого подхода к разработке программ, набор функций может меняться в ходе разработки, а значит, стоимость работ имеет динамику по ходу разработки.

Поэтому самое первое, что нужно сделать, заказчику - это сконцентрировать внимание на составлении пожеланий к тому, что должна делать программа.

Эти пожелания - пользовательские истории нужно записать на карточки и назначить им приоритет.

При этом заказчик не сможет принять бизнес-решения, пока не узнает стоимость реализации того или иного пожелания.

Подразумевается, что объем трудозатрат (стоимость реализации пожеланий) оценивает разработчик.

Но для формирования вызывающей доверие у заказчика оценки требуется диалог и консультации.

Поэтому заказчик должен участвовать ***в фазе исследований*** в самом начале разработки.

После такого участия, заказчик будет хорошо ориентироваться в требованиях к программе и гибко управлять разработкой в ходе

- ***игры в планирование***,
- ***планирования итераций***
- ***написания приемочных тестов***.

Игра в планирование, планирование итераций и написание приемочных тестов – ***это способы принятия бизнес-решений.***

Таким образом, заказчик:

- Знает ЧТО делать
- Знает приоритеты задач
- Желает получить наиболее подходящую ему программу (программную систему)
- НЕ принимает технологических решений
- НЕ задает сроки
- Верит разработчикам

Заказчик имеет право:

- знать план проекта; что может быть выполнено, когда и за какую цену

- контролировать ход проекта, запуская работающую систему, на которой успешно выполняются все тесты
- в любой момент времени запросить изменение функциональности или изменить приоритет задачам
- быть уведомленным об изменении расписания проекта заранее, с тем чтобы успеть решить насколько можно уменьшить объем работ, чтобы успеть вписаться в срок.
- в любой момент закрыть проект, в этом случае должна быть получена работающая система с частью функциональности

Заказчик обязан:

- быть регулярно доступен разработчику с самого начала проекта, быть в команде «по существу»
- знать перечень и приоритеты реализуемых задач программирования;
- понимать нужды бизнеса;
- быть полномочным представителем заказа на разработку (говорить от первого лица (одним голосом));
- согласовывать противоречивые требования к программ;
- разрабатывать приемочные тесты до начала функциональной реализации.

Кто может выступать в качестве заказчиков?

- Представители организаций, где будет использоваться программа,
- Спонсоры организаций, где будет применяться программа,
- Менеджеры

Состав команды разработчиков включает следующих ролевых исполнителей (не в определенный период времени, а в течение всей разработки)

- «Рассказчики»
- «Приемщики»
- «Золотовладельцы»
- «Планировщики»
- «Большой босс»

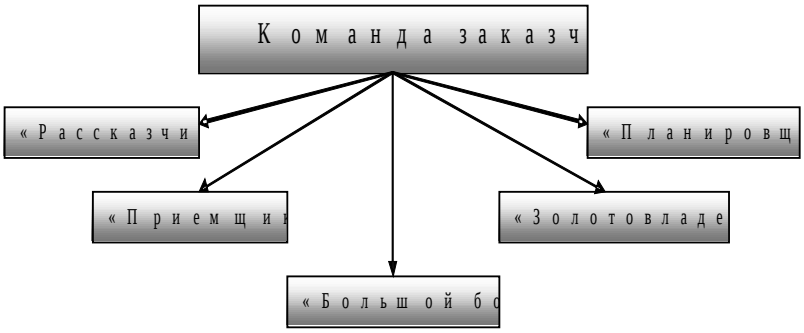


Рис. 48.

«Рассказчики» (storytellers) - это члены команды заказчиков, которые обладают квалификацией в данной предметной области. Они формулируют ту задачу, которую будет решать будущая программная система.

Они пишут «истории пользователей», к которым обращаются разработчики для прояснения требований к системе. В конечном счете рассказчики несут ответственность за правильность решений с точки зрения требований.

Рассказчики обеспечивают продвижение команды разработчиков к версии, готовой к выпуску.

Иногда «рассказчиков» называют «gold donor»- донорами золота, потому что ими могут быть потенциальные ПОЛЬЗОВАТЕЛИ новой системы, которые ожидают ВЫГОДУ от использования этой системы в будущем.

«Приемщики» (acceptors) – это рассказчики или лица, действующие от их имени, которые проверяют программу по приемочным тестам для каждой версии.

Приемщики разрабатывают приемочные тесты.

Наличие приемщиков не освобождает рассказчиков от обязанности знать содержимое каждой версии. По сути приемщики обеспечивают рассказчикам соответствие каждой версии их ожиданиям.

«Золотовладельцы» (gold owners) обеспечивают ресурсы для проекта, проще говоря, финансируют проект. Именно они оценивают стоимость проекта.

«Планировщики» (planners) - это те люди, которые анализируют потребности выпуска версий.

Они составляют расписание выпуска версий сообразно потребностям потенциальных пользователей.

В команде разработчиков аналогичную роль играет так называемый контролер (tracker)

«***Большой босс»*** (big boss) - это человек, который занимается поддержкой системы и отвечающей за нее на всех уровнях. Он работает как бы на обе команды – и заказчиков, и разработчиков, хотя внешне все работают на него.

Часто это лицо одновременно является «золотовладельцем».

Если это не так, и «золотовладелец» работает на «босса», он должен позаботиться о вознаграждении себя и «рассказчиков» за участие в проекте.

Деятельность команды программистов

Хотя каждый разработчик достаточно эгоистичен, имеются роли, которые должны исполнять разработчики для общей пользы.

Разработчики могут играть в команде следующие роли, получая от этого удовольствие и доходы:

- «инструктор»,
- «дипломат»,
- «архитектор»,
- «контролер»,
- «технический писатель»

«Инструктор», или «Тренер» (coach)) - это опытный член команды разработчиков.

Он принимает решения о существе необходимой адаптации процесса разработки и подсказывает другим членам команды как это сделать. За инструктором остается последнее слово в кризисных ситуациях. Его опыт и навыки - залог выполнения проекта на должном уровне без лишних трудностей.

«Контролер или планировщик» - (tracker) обеспечивает согласование оценок проекта с требованиями заказчика.

1. Он следит за сроками выполнения поставленных задач и обсуждает изменения в графике с планировщиком в команде заказчика.
2. Считают, что планировщик = это помощник разработчика в команде заказчика.
3. Контролер обязан своевременно известить разработчиков обо всех изменениях в проекте.
4. Фактически контролер ближе к боссу, чем другие члены команды.
5. Он не работает над программным кодом, но выставляет «оценки» программистам.
6. Он не посещает ежедневные собрания «стоя».
7. Его задача –обеспечить обратную связь с заказчиком, фиксировать в форме отчета историю развития проекта и косвенно участвовать в управлении разработкой.
8. Необходимо, чтобы члены команды доверяли контролеру и его оценкам проделанной ими работы.

«Дипломат» (faciliator) – это член команды разработчиков, обладающий лучшими, чем другие, способностями общаться с людьми.

Его задача – облегчать и поддерживать приемлемые отношения между двумя командами (заказчиками и разработчиками), в особенности при совместных собраниях, посвященных планированию.

«Архитектор» (architects) – создают и перерабатывают архитектуру проекта по мере необходимости.

У архитектора две задачи:

Первая задача - написать минимальное количество контрольных примеров для проверки функций программы.

Вторая задача выявлять запутанные и усложненные фрагменты программного кода на каждой итерации.

«Технический писатель» - пишет документацию.

Обычно в команду входят

- Собственно разработчики (Developer)
- Тренер (Инструктор)
- Технический писатель(Technical Writer)

При этом тренер совмещает функции контролера и дипломата, а собственно разработчики – это программисты, совмещающие проектирование архитектуры с программированием, включая тестирование.

Команда разработчиков:

- Знает КАК делать
- Желает работать максимально эффективно и качественно
- Задает сроки выполнения задач
- НЕ принимает бизнес -решений
- Верит заказчику

Разработчик имеет право:

- знать, что необходимо делать, требования и приоритеты задач.
- не создавать наперед концепцию всей разрабатываемой программы (программной системы).
- сказать, как долго будут реализовываться требования, а также изменить первоначальные оценки по мере накопления опыта.
- принять ответственность, а не получить ее в безусловном порядке.
- постоянно делать качественную продукцию
- на приятную, интересную работу.

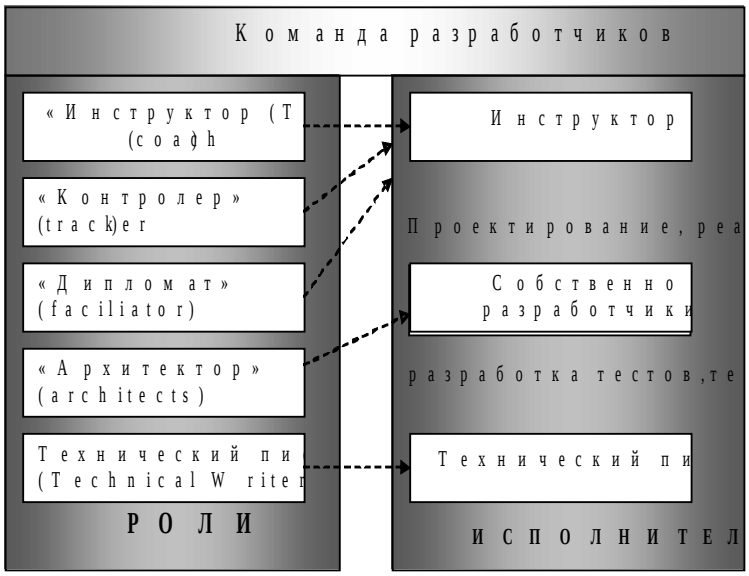


Рис. 49.

Структура идеального XP-процесса

Идеальный XP-процесс используют как эталон теоретически безупречного экстремального программирования.

Основным структурным элементом идеального XP-процесса является **XP-реализация**, в которую многократно вкладывается базовый элемент **XP-итерация**

Макроструктура идеального XP-процесса:

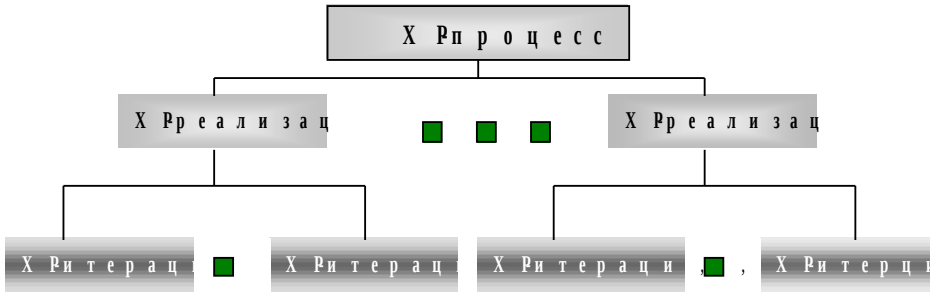


Рис. 50.

Каждая реализация и итерация имеет три фазы развития: исследование, блокировка, развитие:

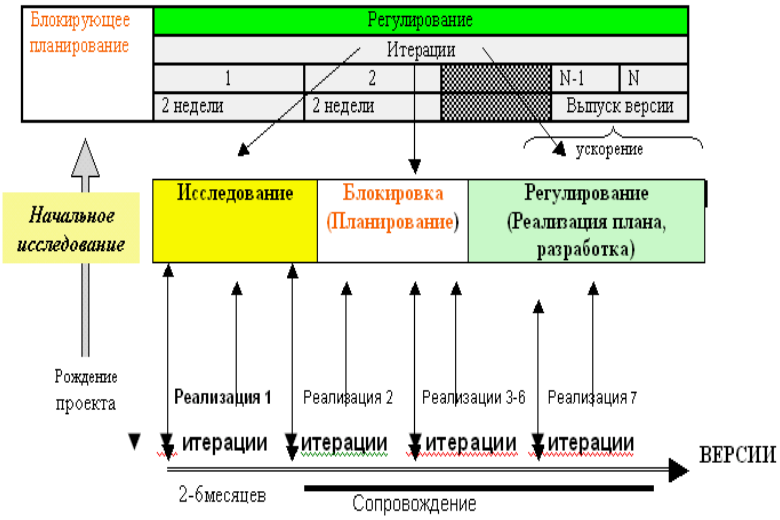


Рис. 51.

Исследование (action) — это поиск новых требований (историй, задач), которые, должна выполнять система. Фаза исследования, с которой начинается любая реализация и итерация должна обеспечить принятие решения о продолжении работы.

Блокировка (commitment) — выбор для реализации конкретного множества из всех возможных требований (иными словами, планирование).

Регулирование (steering) — проведение разработки, воплощение плана

Первая реализация самая трудная, здесь нужно пройти путь от неясных требований к программному средству промышленного качества в среднем за три месяца.

Продолжительность первой реализации в XP – процессе ограничивается интервалом от 2 до 6 месяцев.

Продолжительность остальных реализаций — около двух месяцев.

Первая реализация дает первую версию системы и система начинает использоваться по назначению.

Остальные реализации проводят в рамках разработки, которая имеет задачи обычного сопровождения: устранения ошибок и внесения изменений, но с одним отличием - возможностью расширения функций.

Каждая итерация продолжается приблизительно две недели, за исключением тех, которые относят к поздней стадии реализации – выпуску версии («запуску в производство»). В это время темп итерации ускоряется.

Численность группы разработчиков не должна быть много больше 10 человек.

Структура элемента реализации идеального XP–процесса:

Рис. 52.

<u>Пользовательские истории</u>. и <u>требования к программе</u>

Каждая история занимает несколько предложений в терминах заказчика, которые, обычно записывают на карточках.

Они заменяют традиционную детальную спецификацию требований только теми, которые достаточны для оценки времени их реализации.

Более детальное описание требований следует добиваться в ходе реализации истории.

Каждая история получает оценку идеальной длительности — одну, две или три недели разработки - это время, за которое история может быть реализована при отсутствии других работ.

Если срок превышает три недели, историю следует разбить на несколько частей.

При длительности менее недели нужно объединить несколько историй в одну.

<u>Неопределенные оценки и выброс</u>.

Если разработчики не знают, как оценить историю, они создают выброс - решение, содержащее ответ на трудные вопросы.

Выброс — быстрое минимальное решение, выполняемое в черновом коде и впоследствии выбрасываемое. Результат выброса — знание, достаточное для оценивания истории.

Архитектурный выброс и метафора системы .

Архитектурный выброс- это прототип программы, который в дальнейшем не используется. Он нужен только для выявления замысла разработки.

Итогом архитектурного выброса является создание метафоры системы.

Метафора системы определяет ее состав и именование элементов в единой аналогии с известными понятиями.

Это важно для понимания всеми участниками общего замысла разработки программы.

Планирование реализации.

Планирование устанавливает правила вовлечения заинтересованных сторон в реализацию проекта. К ним относят разработчиков, заказчиков и менеджеров .

Главный итог- **план выпуска версий для всей реализаци**и.

План реализации определяет даты выпуска версий и пользовательские истории, которые будут реализованы в каждой их них.

Итерации детально планируют позже - перед началом ее реализации.

Важнейшим ограничением планирования являются сроки реализации, которые вытекают из полученных оценок времени реализации историй и пожеланий заказчика.

Структура итерации:

Рис. 53.

Итерация обеспечивает проекту необходимую динамику и постоянную готовность к внесению изменений. Поэтому в начале итерации проводят не долгосрочное (стратегическое), а краткосрочное (тактическое) планирование.

Пользовательским историям и соответствующим тестам сопоставляют задачи программирования.

Описания задач, как и истории, записывают на карточках (иногда на обратной стороне).

Главная цель планирования итерации - выработка плана решения программных задач в срок от одной до двух-трех недель.

К очередным историям добавляют те, которые не прошли тесты приемки предыдущей итерации и требуют доработки.

Истории сортируют (ранжируют) в порядке их значимости с точки зрения заказчика.

Это дает хороший эффект оправдания ожиданий заказчика от каждой итерации.

Фактически детальный план итерации представляют записанные на карточках программные задачи, которые ранжированы в соответствии с ранжированием пользовательских историй.

Стратегическое и тактическое планирование

Планирование в XP-разработке поглощает традиционный анализ требований и проектирование.

Оно включает следующие виды деятельности:

- Игра в планирование (предварительное планирование)
- Стратегическое планирование (планирование версии);
- Тактическое планирование (планирование итерации).

Каждый элемент идеальногоXP-процесса (итерация и реализация) включают планирование в качестве блокирующего фактора разработки с целью принятия решения на множестве альтернатив.

Игра в планирование

Игра в планирование – это первоначальный обмен информацией между заказчиками и разработчиками, в процессе которого фиксируют пользовательские истории.

Игра в планирование предполагает описание программы

- с помощью карточки представления о программе,
- пользовательских историй,
- метафоры,
- предварительного плана работы.

Стратегическое планирование включает разработку плана выпуска версии с разбивкой на итерации.

Тактическое планирование состоит в разработке плана каждой итерации день за днем.

Необходимым условием проведения стратегического и тактического планирования является предоставление планировщикам оценок трудоемкости задач историй пользователей.

Планирование в XP — это скорее искусство, **чем** точная наука.

Разработчики вправе ошибаться по поводу требуемого для разработки времени.

Но известное стремление угодить заказчику, может привести к негативным результатам.

Чтобы свести к минимуму риск отсутствия конструктивного общения заказчиков и разработчиков, экстремальное программирование предоставляет им определенные права.

Например, хотя разработчикам не всегда удается делать **верные оценки, заказчики все равно имеют право** составить план, который впоследствии может быть изменен.

Участвуя в планировании, заказчик имеет следующие права:

- Право на составление общего плана, чтобы знать , что будет выполнено, когда и с какими затратами.
- Право получать максимальный результат из каждой недели программирования.
- Право поменять свое мнение, заменить какие-либо функции и изменить приоритеты , не неся при этом чрезмерных дополнительных издержек

Разработчик имеет право:

- Знать, что требуется сделать и каковы приоритеты.
- Самому брать на себя обязательства, а не следовать жестким предписаниям.
- Изменять свои оценки.

Основная цель игры в планирование - формирование карточек с пользовательскими историями, и предварительного плана работы с отражением следующих вопросов:

1. Что должна делать программа (область действия)?
2. Какие аспекты наиболее важны, что можно убрать и что отложить (приоритеты)?
3. Что должна вмещать каждая версия (содержимое версий)?
4. Каковы сроки выпуска версий (предварительные даты выпуска версий)?
5. Сколько времени понадобиться на реализацию пользовательских историй?
6. Каковы затраты на инструментальные средства разработки?
7. Каким образом будет разрабатываться программа (риски разработки, организация команды, задачи программирования).

   Решение вопросов 1-4 является прерогативой заказчика,
   а вопросы 5-7 – прерогативой разработчика.
   Таким образом, можно констатировать, что
   в XP-программировании требования к программе – это диалог, а
   не документ.

Термин «игра в планирование» выглядит довольно несерьезным. Вместо него можно использовать какой-нибудь другой термин. Например, ***совещание по планированию проекта.***

Важно понимать, что игра в планирование – это самый первый шаг, который используют для формирования плана проекта.

В ходе работы над проектом, игра в планирование может использоваться **несколько раз** для пересмотра и преобразования первоначального плана**.**

Игра в планирование проходит достаточно просто:

1. **Заказчик записывает пожелания-истории на карточках.**
2. **Разработчик записывает па каждой карточке оценку трудозатрат. Для оценки трудозатрат используются специальные единицы измерения - пункты (количество полных 40-часовых недель для реализации пожелания-истории).**
3. **Исходя из некоторого заранее определенного уровня ресурсов, разработчик определяет, сколько историй может быть выполнено за одну итерацию длительностью N недель (от одной до четырех включительно).**
4. **Заказчик раскладывает карточки с пожеланиями в стопки в том порядке, в котором он хотел бы, чтобы они были реализованы. Для удобства карточки могут объединяться в пачки, содержащие версии и вложенные в них итерации.**

   Это краткое описание. Более подробно можно выделить следующие аспекты.

   Заказчик начинает рассказывать, что должна делать разрабатываемая программа.

Пока он рассказывает, необходимо записывать его пожелания (story) на карточки. (Возможно, перед тем, как пожелание в окончательном виде будет записано на карточке, в течение некоторого времени используется промежуточный носитель (доска, блокнот и.т.д))

Перед тем как завершить игру, заказчик должен отсортировать карточки.

Пожелания заказчика можно хранить также в электронной форме, но лучше делать электронную версию после окончания игры в планирование.

Пока заказчик формулирует свои пожелания, разработчики могут задавать уточняющие вопросы. Заказчик отвечает на них.

Планирование версий

Между начальным планированием и планированием выпуска версии продукта существует несколько отличий

- Начальное планирование зачастую выполняется **еще** до того, как будет сформирована команда разработчиков, которая будет работать над продуктом.
- В начальном планировании может участвовать всего один разработчик и один заказчик, но при этом допускается посторонняя помощь.
- Начальное планирование рекомендуется выполнять как можно раньше, тогда, когда проект только созревает, когда он находится в стадии начального обдумывания.
- Планирование выпуска версий осуществляют в присутствии всех разработчиков и всех представителей заказчика.
- Планирование выпуска необходимо выполнять тогда, когда изучены все основные подходы к решению проблемы, когда есть необходимые ресурсы, чтобы немедленно начать работу.

Помимо этого разница состоит в уровне детализации:

- в процессе начального планирования делают лишь приблизительные оценки важности и трудоемкости выполнения работ самого первого уровня детализации.
- Например, работы могут быть оценены как **обязательные, очень желательные, желательные, но не обязательные**.
- При этом все оценки делят на три категории: с **малым риском, со средним риском и с высоким риском**. Рискованность работ связывают с тем ущербом, к которому может привести невыполнение этих работ.
- Вначале следует реализовать работы из категории обязательных, а среди них первоочередными будут работы с наибольшим риском.

В процессе планирования версий оценки уточняют, но не тратя на одну оценку более одного дня.

При этом оценки тоже могут быть не точными, но все же более реалистичными.

Для уточнения всех оценок перед планированием версий желательно проводить *фазу исследований. Продолжительность фазы исследования складывается из времени, необходимого для уточнения всех оценок с некоторым временным запасом.*

Если заказчик не *хочет* тратить время на уточнение оценок, нужно постараться все - же провести ее, склонив к этому заказчика, увеличив риски существующих оценок (например, в два раза) .

Помимо уточнения первоначальных оценок работ планирование версий включает :

1. Оценку скорости (производительности) работы над версией,
2. Оценку стоимости версии (через трудозатраты)
3. Определение приоритетов задач
4. Составление плана выпуска версии.

Оценка скорости работы над версией

Скорость проекта – это количество историй в одной итерации.

Быстродействием команды называют количество идеальных недель в пунктах историй, то есть количество историй, которые реализуются за идеальную неделю.

Зная свои реальные возможности, при планировании версии разработчики могут уточнить время разработки и провести оценку скорости выпуска версии, которая влияет на стоимость версии.

Для оценки производительности на первой итерации версии можно использовать показатель, численно равный количеству человеко-дней в итерации, поделенному на три.

Это будет означать, что затраты на каждую историю будут в три раза больше, чем предполагалось вначале.

После каждой итерации эту величину следует либо корректировать, либо заменить другим показателем. Напомним, что производительность определяет и контролирует разработчик, играющий роль контролера.

Стоимость версии определяется произведением стоимости одного человека-часа на количество человеко-часов в версии.

План выпуска версий устанавливает

- перечень работ и их исполнителей,
- очередность
- сроки выполнения работ.

Ключевыми факторами планирования являются

- пользовательские истории с оценками,
- ресурсы для разработки
- скорость работы (производительность).

При составлении плана можно использовать как обычные таблицы и графики, так и специальные методы сетевого планирования.

Планирование итераций

Каждая версия должна пройти набор приемочных тестов пока не будет признана удовлетворительной.

Как только команда разработчиков «увидит» версию, например, получит результаты очередного тестирования, может появиться необходимость внести небольшие изменения.

Создание лучших версий должно стать для разработчиков частью их работы. **Поэтому при разработке версий применяют итерации**.

Истории пользователя расписываются как часть этих итераций таким образом, чтобы улучшить версию, созданную на прошлой итерации.

Суть планирования итераций заключается в делении версии на серию коротких, улучшающих версию итераций.

Продолжительность итераций определяется многими факторами, в том числе

- уровнем автоматизации разработки,
- уровнем организации труда,
- средой разработки,
- особенностями команд разработчиков и заказчиков.

В больших проектах итерации могут быть от двух до шести «идеальных» недель.

Как правило, итерация не превышает четырех недель.

После определения продолжительности итерации составляют расписание историй пользователя.

Для каждой итерации устанавливают так называемый бюджет.

Единицей измерения бюджета является пункт – идеальная сорокачасовая рабочая неделя.

Численно бюджет определяют как произведение продолжительности итерации (в идеальных неделях) на число разработчиков и деленное на коэффициент нагрузки (количество реальных недель в одной идеальной неделе):

$$\text{Бюджет итера} = \frac{\text{Продолжительность итерации} \times \text{Количество разраб}}{\text{Количество реальных недель в одно}}$$

Рис. 54.

Планирование итераций выполняется на собрании групп заказчиков и разработчиков.

Чтобы расписать итерации, можно использовать обычную таблицу: итерация становится столбцом, а истории пользователя – строками.

Для удобства обозрения такую таблицу изображают либо на доске, либо на другой подходящей горизонтальной поверхности (стена, стол).

Столбцы и строки таблицы наглядно отражают связи историй и итераций, отслеживание которых входит в задачу планирования.

История не может быть включена в итерацию, если она зависит от истории, которая еще не включена в расписание.

Команда разработчиков играет в планировании активную роль.

После того, как истории пользователя сгруппированы по итерациям, разработчики проверяют применимость к ним имеющихся оценок.

Результаты собрания должны быть зафиксированы в результирующей электронной таблице.

Каждая история пользователя, попадающая в версию, должна иметь номер итерации.

Этот номер может измениться, если увеличивается или уменьшается скорость проекта.

Первая итерация — особенная в экстремальном программировании.

Она определяет ядро программы, на котором будут построены все остальные итерации.

Выпуск этой части программы служит заказчикам подтверждением возможностей исполнителей.

Разработчики и заказчики должны убедиться, что экстремальное программирование работает.

В ядре должно содержаться несколько самых важных историй, сосредоточенных вокруг основных функций.

Но трудоемкость этих историй не должна выходить за рамки выделенного бюджета итерации.

Цель каждой новой итерации — построение функции на той основе, которая заложена на первой итерации.

Последним шагом планирования итерации является определение календарных дат (начала и конца) итерации.

Реальными результатами планирования итераций являются:

- составление расписаний,
- разбиение историй на задачи,
- закрепление задач за разработчиками,
- разбиение разработчиков на пары.

Основные принципы планирования:

1. Очередность включения историй в итерации определяется важностью историй
2. Наполнение итерации историями определяется ее бюджетом (а не наоборот!). Поэтому время итерации является самым стабильным элементом плана.
3. Следующая итерация планируется тогда, когда исчерпан бюджет предыдущей.
4. План итерации может меняться. Главное составить его так, чтобы наиболее важные истории попали в наиболее ранние итерации.
5. В ходе разработки нельзя менять календарные даты итераций. Это косвенно означает, что даты выпуска версий переносить нельзя.
6. Нереализованную к дате выпуска версии пользовательскую историю можно перенести в другую итерацию. Но вначале должны быть реализованы те истории, которые представляют наибольший риск в версии.
7. Каждая итерация должна порождать версию, а серия итераций - к выпуску версии, представляющей функционирующую по назначению программу.

8. Фактически в проекте XP существует два типа планирующих расписаний: сроки для версий и сроки для итераций.

Перспективы развития технологий экстремального программирования.

Автоматизация программирования.

Главная цель автоматизации — избавить человечество от рутины, передать ее компьютерам, освободить человеку время для выполнения тех задач, которые передать машинам невозможно. Именно это является ключевым направлением эволюции компьютерных технологий, которое определяет бизнес-стратегии успешных высокотехнологических компаний

Несмотря на постоянное повышение эффективности компьютеров продуктивность человеческого труда увеличивается весьма скромными темпами, особенно это касается производительности труда программистов.

Ситуацию могут исправить такие методологии как

- INTSPEI P-Modeling Framework,
- обратная семантическая трассировка
- и др.,

позволяющие оптимизировать этапы процесса создания программного обеспечения, которые трудно поддаются автоматизации.

В компаниях, производящих программное обеспечение, особое внимание должно быть уделено продуктивности труда инженеров — к этому подталкивают не только основанные на макропоказателях экономические выкладки, но и банальная нехватка программистов на рынке труда.

Эффективность труда инженеров — ключевое конкурентное преимущество, поэтому новые методологии разработки программного обеспечения появляются сегодня почти так же часто, как появлялись новые языки программирования два десятилетия назад:

- Agile Unified Process (AUP),
- Design-Driven Development (D3),
- Dynamic Systems Development Method (DSDM),
- Test-Driven Development (TDD) и т.п.

Такое разнообразие подтверждает, что ИТ-отрасль еще далека от зрелости.

К примеру, eXtreme Programming — скорая (agile) методология, содержащая набор практик и рекомендаций, помогающих инженерам лучше организовать их работу.

Пожалуй, самой известной методикой XP является «парное программирование» — два инженера совместно работают над задачей, с выполнением которой каждый из них мог бы справиться в одиночку.

С «традиционной» точки зрения, «парное программирование» создает ненужную избыточность и неэффективно использует человеческие ресурсы. Но на практике оно ведет к повышению качества кода и снижению количества последующих переработок, тем самым сокращая общий производственный цикл.

При парном программировании продуктивность труда повышается не за счет автоматизации или применения сложных алгоритмов, **а за счет оптимизации взаимодействия людей друг с другом.**

Методология INTSPEI P-Modeling Framework (надстройка над существующими методологиями, расширяющая их новыми методиками и механизмами, www.intspei.com) менее известна и касается главным образом рекомендаций по

оптимизации деятельности инженеров.

Например, **«бессловесное моделирование»**, согласно которому аналитики и проектировщики проводят сессию моделирования и создают архитектуру будущей системы, не используя «человеческие» языки — для взаимодействия с коллегами может применяться **только UML** или иной язык моделирования.

С «традиционной» точки зрения, «бессловесное моделирование» должно вести к потере информации, но в действительности оно повышает качество создаваемой модели и понижает количество проявляющих себя в последующем архитектурных проблем, тем самым сокращая общий производственный цикл.

И снова продуктивность труда повышается не за счет автоматизации, **а за счет применения новых методов организации труда людей**.

Ранняя диагностика архитектурных ошибок

Начальные стадии (анализ и проектирование) больших проектов являются чрезвычайно ответственными — именно здесь принимаются наиболее важные решения и допускаются ошибки, которые наиболее дорого обходятся впоследствии.

Затем, по мере развития проекта, стоимость ошибки снижается. В то же время уровень контроля качества на начальной стадии является минимальным и повышается в процессе продвижения процесса разработки.

К полноценному тестированию приступают только на поздних этапах проекта — тогда, когда уже имеется исполнимый код.

Результатом такого «эффекта ножниц» часто становится несвоевременная (поздняя) диагностика архитектурных ошибок и последующая дорогостоящая переработка всей системы либо ее отдельных частей.

Для решения проблемы задержек диагностики есть два пути.

1. Первый — сокращение длительности итераций. Время между моментом внесения ошибки и моментом ее выявления не может быть больше, чем длительность итерации. Сокращая итерации, мы сокращаем максимальную возможную задержку диагностики ошибок. Именно по этому пути — частых итераций — идут скорые методологии. Однако этот путь неприемлем для больших проектов, а также для проектов с повышенными требованиями к качеству (медицинские, аэрокосмические, военные приложения).
2. Второй путь — включение дополнительных методов контроля качества на каждой фазе процесса создания программного обеспечения. Теоретически с их помощью мы могли бы выявлять и устранять архитектурные ошибки сразу в момент их возникновения, не позволяя ошибкам распространяться на последующие фазы производственного процесса. В этом случае можно было бы говорить не столько о выявлении ошибок, сколько об их предотвращении.

Практически же существовавшие до сих пор методы не позволяли делать это эффективно. В результате более чем шести лет экспериментально-исследовательских работ сегодня INTSPEI располагает технологией раннего выявления критических ошибок, в основе которой лежит метод обратной семантической трассировки.

Метод обратной семантической трассировки

Представим себе гипотетическую ситуацию. Предприятие планирует выйти на рынок Японии, и у него нет переводчиков с японского, но имеется местная компания-партнер, сотрудники которой не говорят по-русски.

Предприятие нанимает внешнюю команду лингвистов, которые переводят

маркетинговые материалы.

После того как они заканчивают работу, результаты перевода отсылаются на проверку партнеру в Страну восходящего солнца.

В ответ предприятие получает сообщение, в котором на ломаном русском сообщается, что полученные тексты действительно написаны на японском языке.

- Как оценить качество перевода?
- Как проверить, что переводчики всю информацию передали правильно и ничего не исказили?
- Как избежать эффекта «испорченного телефона»?

От ответов на эти вопросы зависит успех запускаемой в Японии массовой рекламной кампании.

Обычный ответ — перевести назад с японского на русский, для чего привлечь новую команду лингвистов, которая не знакома с первоначальными русскими версиями текстов.

Таким образом, для процесса перевода существует простой метод проверки его качества путем **«обратной инженерии»** результатов перевода с последующим сравнением оригинального и восстановленного текстов.

Подобный же метод можно применить и при разработке программного обеспечения.

Действительно, процесс создания программ можно рассматривать как последовательность «переводов» с одного языка на другой.

1. Аналитик «переводит» с человеческого языка на язык формального описания бизнес-процессов,
2. проектировщик «переводит» полученный текст на язык описания архитектуры,
3. затем программист «переводит» его на язык программирования,
4. и наконец компилятор «переводит» результат работы программиста на язык машинных кодов.

Каждый из этих шагов имеет две составляющие:

- трансляция информации с предыдущего шага на последующий;
- понижение уровня абстракции, детализация транслируемой информации и принятие технических решений.

Очевидно, что проверка путем обратного «перевода» поможет выявить не все ошибки, а только те, которые относятся к первой составляющей. Однако и это немало, тем более что большинство проблем при разработке и внедрении программного обеспечения связано не с неверными техническими решениями, а с ошибками коммуникации.

Упомянутый метод получил название **обратной семантической трассировки (ОСТ)** и в общем виде применим к произвольным шагам информационных процессов, имеющим своей целью преобразование информации из входных артефактов в выходные. Метод позволяет:

- оценить качество трансляции информации, преобразованной в результате осуществления шага процесса;
- принять решение о необходимости коррекции входных и/или результирующих артефактов;
- принять решение о необходимости повторного осуществления шага процесса или о возможности перехода к следующему шагу.

Применение этого метода совместно с «традиционными» методологиями позволяет осуществлять сквозной контроль информационных потоков в процессе создания программного обеспечения.

Использование термина «семантическая» в названии подчеркивает тот факт, что при сравнении оригинальной и восстановленной версий «текстов» необходимо анализировать их сходство и различие на содержательном, семантическом уровне, что

пока автоматизировать невозможно.

В основе бесплатного продукта INTSPEI P-Modeling Framework лежат существующие методологии, дополненные методиками, направленными на повышение эффективности труда инженеров и предотвращение критических ошибок.

Первая версия INTSPEI P-Modeling Framework была создана в сотрудничестве со специалистами из IBM и Microsoft и интегрируется в IBM Rational Unified Process (RUP), Open Unified Process (OpenUP) и Microsoft Solutions Framework (MSF).

Каждый из этих продуктов представляет собой интерактивный справочник, содержащий детальное описание соответствующего производственного процесса, предусмотренных им фаз, этапов, задач, шагов, процедур; производственных ролей и их зон ответственности; создаваемых и используемых артефактов, их шаблонов и т.п.

Проблема повторного использования кода.

Каждый программист, выполняя свою работу, производит код для повторного использования и применяет код повторного использования.

Любой макрос или библиотека представляют собой ранее произведённый и оптимизированный продукт, применение которого вносит очевидные преимущества в текущую разработку.

Для программирования мы применяем множество доступных инструментов, но создаваемый нами код является наиболее ценным и постоянно модифицируемым набором инструментальных программных средств повторного применения.

Вариант кода, пригодный для применения, должен быть таким, в котором способен разобраться человек, не являющийся автором этого кода. Если такое определение не действует, применение такого кода может привести к перепрограммированию части текущего проекта заново.

Компания IBM начала поставки комплекта шаблонов серверных приложений San Francisco Project.

Пакет San Francisco Project, который к настоящему моменту получили десять компаний-разработчиков ПО, представляет собой набор из тысячи объектно-ориентированных, системно-независимых библиотек классов - своеобразных строительных блоков для создания серверных приложений, работающих на платформах IBM и других производителей.

Первые три шаблона предназначены для создания приложений по

- ведению бухгалтерской отчетности,
- обработке заказов
- и управлению складами,

работающих практически на любых серверных платформах.

San Francisco Project позволяет создавать системно-независимые приложения, экономит усилия разработчиков и, из-за возможности многократного использования одних и тех же фрагментов программ, ускоряет выпуск продуктов.

Благодаря San Francisco компаниям больше не придется каждый раз изобретать велосипед при создании программ, выполняющих общепринятые операции, и можно будет уделять больше внимания разработке уникальных функций.

San Francisco освободит разработчиков от необходимости написания драйверов, API-интерфейсов и прочих низкоуровневых программ.

Генеративное программирование

Наука программирования по применению ранее разработанного кода, являющаяся частью программной инженерии, называется **«Порождающее программирование (Generative programming)»**.

Единственная на данный момент времени переведенная на русский язык

фундаментальная книга К. Чарнецки и У. Айзенекера “Порождающее программирование: методы, инструменты, применение”, имеет оригинальное наименование “Generative Programming: Methods, Tools, and Applications”, поэтому имеет ещё и другое название: **«Генеративное программирование»**.

Идеи порождающего программирования основываются на проектировании и построении порождающих моделей для семейств систем с целью генерирования по этим моделям конкретной системы.

При наличии практического опыта при разработке семейства систем в определенной предметной области (что характерно для прикладного программирования), появляется возможность разрабатывать многократно используемое ПО.

Разработка многократно используемых компонентов позволяет выявить как общности членов семейства, так и наличие существенных параметров изменчивости.

Применение инициаторами выпуска книги в России и авторами перевода синонима “порождающее” к оригинальному термину “генеративное” - не случайное и оправданное решение, связанное с привлечением внимания к одному из важнейших направлений в современном развитии программной инженерии – к автоматизации сборки разрабатываемого программного обеспечения.

Вместе с тем в оригинальном названии книги корнем термина “генеративное программирование” является слово “генератор”. То есть, генеративное программирование предполагает использование генераторов программного кода, которые на основе ранее разработанного кода и с учетом спецификаций требований позволяют автоматизировать сборку ”пилотной” и рабочей версий проекта.

Генеративное программирование, являясь инструментарием архитекторов ПО, предназначено для управления логикой проекта от стадии его разработки и далее, на протяжении жизненного цикла программной системы.

Для применения генеративного программирования архитектору ПО необходимы

- CASE- системы, пригодные для их применения в конкретной предметной области,
- генераторы ПО
- ранее разработанный программный код, пригодный для его повторного применения.

Генераторы ПО

Программная среда, которая осуществляет сборку готового к применению программного продукта, в том числе:

- программной системы,
- компонента,
- класса,
- процедуры
- и тому подобных частей системы

на основе высокоуровневой спецификации, называется “генератор”.

Применение генераторов позволяет

- повышать точность и ясность описаний проектируемых систем за счет использования предметно-ориентированных нотаций, реализуемых при помощи генераторов,
- автоматизировать процесс определения эффективности реализации проектируемой системы
- оптимизировать работу с библиотеками компонентов.

Генераторы реализуют множество различных технологий, включая

1. препроцессоры,
2. метафункции,

3. компиляторы,
4. генерирующие классы и процедуры,
5. генераторы кода CASE-систем (инструментов автоматизированного проектирования и создания программ),
6. трансформационные компоненты
7. и многое другое.

Например, при помощи генератора можно автоматически сгенерировать реализацию программы на машинном языке или в виде байтового кода, исходный код которой был написан на высокоуровневом языке программирования.

Большинство CASE-систем содержат генераторы реализаций графических моделей в виде описаний на языке программирования.

Некоторые системы метапрограммирования позволяют использовать генераторы, имея в качестве входных спецификаций графически специфицируемые конфигурации компонентов в виде предметно-ориентированных нотаций и фрагментов кода на высокоуровневом языке.

Применение архитектурных образцов для проектирования ПО

Несмотря на 30-летний опыт применения термина архитектура программного обеспечения, практическое внедрение архитектурного проектирования ПО продолжается до сих пор и по-прежнему считается новым технологическим направлением в промышленном программировании.

Учитывая компоновочный характер построения проектируемых программных систем, в литературе, посвященной архитектурному проектированию, большое внимание отводится разработкам и внедрению образцов (паттернов) проектирования ПО.

В статье-справочнике Ольги Дубиной “Обзор паттернов проектирования” дается такое определение образцов проектирования ПО “…Любой паттерн проектирования, используемый при разработке информационных систем, представляет собой

- формализованное описание часто встречающейся задачи проектирования,
- удачное решение данной задачи,
- а также рекомендации по применению этого решения в различных ситуациях.
- Кроме того, паттерн проектирования обязательно имеет общеупотребимое наименование.
- Правильно сформулированный паттерн проектирования позволяет, отыскав однажды удачное решение, пользоваться им снова и снова.

Следует подчеркнуть, что важным начальным этапом при работе с паттернами является адекватное моделирование рассматриваемой предметной области. Это является необходимым как для получения должным образом формализованной постановки задачи, так и для выбора подходящих паттернов проектирования…”.

К. Чарнецки и У. Айзенекер считают, что во многих случаях при архитектурном проектировании крайне полезно проводить периодическую сортировку паттернов проектирования (architectural patterns) с целью получения новых вариантов сортировки.

Полученный таким образом новый архитектурный образец должен соответствовать определенному набору требований, иметь описание в виде документации.

В качестве иллюстрации К. Чарнецки и У. Айзенекер приводят перечень примеров архитектурных образцов:

- Уровневый образец. Сортировка по группам подзадач, каждая из групп находится на определенном уровне абстракции;
- Образец каналов и фильтров. Схема обработки потока данных предполагающая, что некоторое количество этапов обработки инкапсулировано в компоненты фильтрации. Данные передаются по

каналам между смежными фильтрами, рекомпоновка фильтров позволяет собирать связанные системы или обеспечивать сходное поведение систем;

- Образец “классной доски”. Схема, в которой осуществляется объединение знаний нескольких специализированных подсистем; что позволяет находить частное или приближенное решение недетерминированной задачи;
- Образец-посредник. Схема, в которой разъединенные компоненты взаимодействуют посредством удаленных служб. Необходимо наличие компонента-посредника, обеспечивающего координацию взаимодействия и передачу результатов и исключений;
- Образец модель—представление—контроллер. Разложение системы на три компонента: модель с базовыми функциональными возможности и данными, представления для отображения информации пользователю, и контроллеры для обработки данных пользователя. Непротиворечивость данных пользовательского интерфейса и модели обеспечиваются механизмом распространения изменений.
- Образец микроядра. Схема, в которой базовое функциональное ядро отделено от функций и деталей, выполняемых по индивидуальным заказам потребителей. Кроме того, микроядро, к которому подключаются эти расширения, организует их взаимодействие.

Михаил Ксензов в статье "Рефакторинг архитектуры программного обеспечения: выделение слоев" особо выделяет паттерны архитектурного рефакторинга, которые применяются к компонентам архитектуры на примере паттерна выделения слоев.

Концепция слоев, особо выделяемая Михаилом Ксензовым, – это одна из общеупотребительных моделей, используемых разработчиками программного обеспечения для разделения сложных систем на более простые части.

В архитектурах компьютерных систем, например, различают

- слои кода на языке программирования,
- функций операционной системы,
- драйверов устройств,
- наборов инструкций центрального процессора
- и внутренней логики микросхем.

Например, в среде сетевого взаимодействия протокол FTP6 работает на основе протокола TCP7, который, в свою очередь, функционирует "поверх" протокола IP8, расположенного "над" протоколом Ethernet9 и так далее.

Развитие методов объектно-ориентированного программирования повлияло на использование шаблонов метапрограммирования. Использование архитектурных образцов в виде шаблонов метапрограммирования представляют собой практические примеры внедрения генераторов в библиотеки C++.

Интенциальное программирование

Автоматизация применения порождающего программирования сопровождается исследованиями по созданию специальных систем (сред) метапрограммирования, которые предлагают более широкую поддержку порождающего программирования.

Ярким и пока единственным примером реализаций таких систем является система IP, разработанная Чарльзом (Каролем) Симони (Charles (Karoly) Simonyi) в период его работы в корпорации Microsoft.

Работа в среде метапрограммирования IP (Intentional Programming) иногда называется **ментальным или интенциальным программированием**.

Данное направление также раразрабатывается в компании JetBrains, генеральным

директором которой является известный специалист и создатель IntelliJ IDE for Java, Сергей Дмитриев.

Компьютерные журналисты называют работы Сергея фабрикой кодогенераторов, особенно в связи с созданием в его компании технологии Meta Programming System (MPS).

Автоматное программирование

Наиболее известной Российской разработкой в области автоматизации программной инженерии является метод проектирования и реализации реактивных объектно-ориентированных программ с явным выделением состояний.

Для поддержки метода разработано инструментальное средство UniMod, являющееся исполняемым языком моделирования UML для генерации ПО.

Метод основан на использовании автоматного программирования (SWITCH-технология) и UML-нотации.

Базирующееся на этом методе инструментальное средство программирования UniMod, является встраиваемым модулем для платформы Eclipse.

Внастоящее время язык моделирования UML применяется, в основном, как язык спецификации моделей систем.

Существующие UML-средства позволяют строить различные диаграммы и автоматически создавать по диаграмме классов скелет кода на языках программирования, а так же предоставляют возможность автоматически генерировать код поведения программы по диаграммам состояний.

Однако известные инструменты не позволяют в полной мере эффективно связывать скелет кода с моделью поведения, которую можно описывать с помощью диаграмм

- состояний,
- деятельностей,
- кооперации
- или последовательностей.

Отсутствие однозначной операционной семантики при традиционном написании программ приводит к различию описания поведения в модели и в программе, а также к произвольной интерпретации программистами поведенческих диаграмм. Описание поведения в модели часто носит неформальный характер.

Появление операционной семантики в языке UML идентифицирует однозначность понимания диаграмм участниками проектирования ПО и позволит создать исполняемый UML, а генерация кода при этом может, в частности, выполняться непосредственно при интерпретации описанной модели.

Особенность реализованного в инструментальном средстве UniMod метода состоит в том, что проектирование программы выполняется так же, как проводится автоматизация технологических процессов для данной предметной области.

Авторы инструментального средства UniMod, сохранив автоматный подход, реализовали UML-нотацию при построении диаграмм в рамках SWITCH-технологии.

Эвристическое программирование.

Эвристические алгоритмы - алгоритмы, сужающие пространство перебора за счёт поиска решения с учётом особенностей задачи на основе эвристик, которые позволяют ускорить процесс решения задачи.

Значительный интерес к их исследованию возник в связи с возможностью решения ряда задач (распознавание объектов, доказательство теорем и т. д.), в которых человек не может дать точный алгоритм решения, с помощью технических устройств.

Назначением *Эвристики* является построение моделей процесса решения какой-либо новой задачи.

Существуют следующие типы таких моделей:

- модель слепого поиска, которая опирается на так называемый метод проб и ошибок;
- лабиринтная модель, в которой решаемая задача рассматривается как лабиринт, а процесс поиска решения — как блуждание по лабиринту;
- структурно-семантическая модель, которая считается в настоящее время наиболее содержательной и которая отражает семантические отношения между объектами, составляющими область задачи.

Использование эвристических методов (эвристик) сокращает время решения задачи по сравнению с методом полного ненаправленного перебора возможных альтернатив; получаемые решения не являются, как правило, наилучшими, а относятся лишь к множеству допустимых решений; применение эвристических методов не всегда обеспечивает достижение поставленной цели.

Эвристика понимается и как совокупность присущих человеку механизмов, с помощью которых порождаются процедуры, направленные на решение творческой задач (например, механизмы установления ситуативных отношений в проблемной ситуации, отсечения неперспективных ветвей в дереве вариантов, формирования опровержений с. помощью контрпримеров и т.п.). Эти механизмы, в совокупности определяющие метатеорию решения творческих задач, универсальны по своему характеру и не зависят от конкретной решаемой проблемы.

Если при обычном программировании программист перекодирует готовый математический метод решения в форму, понятную ЭВМ, то в случае эвристического программирования он пытается формализовать тот интуитивно понимаемый метод решения задачи, которым, по его мнению, пользуется человек при решении подобных задач. Как и эвристические методы, эвристические программы не обеспечивают абсолютного достижения поставленной цели и оптимальность получаемого результата.

При эвристическом программировании могут быть использованы базовые компоненты экспертных систем, хорошо зарекомендовавшие себя на практике (машина логического вывода и подсистема представления знаний).

С помощью этой технологии решаются следующие типы задач:

- *Интерпретирующие системы* предназначены для формирования описания ситуаций по результатам наблюдений или данным, получаемым от различного рода сенсоров. Типичные задачи, решаемые с помощью интерпретирующих систем, — распознавание образов и определение химической структуры вещества.
- *Прогнозирующие системы* предназначены для логического анализа возможных последствий заданных ситуаций или событий. Типичные задачи для экспертных систем этого типа — предсказание погоды и прогноз ситуаций на финансовых рынках.
- *Диагностические системы* предназначены для обнаружения источников неисправностей по результатам наблюдений за поведением контролируемой системы (технической или биологической). В эту категорию входит широкий спектр задач в самых различных предметных областях — медицине, механике, электронике и т.д.
- *Системы проектирования* предназначены для структурного синтеза конфигурации объектов (компонентов проектируемой системы) при заданных ограничениях. Типичными задачами для таких систем является синтез электронных схем, компоновка архитектурных планов, оптимальное размещение объектов в ограниченном пространстве.
- *Системы планирования* предназначены для подготовки планов проведения последовательности операций, приводящей к заданной цели. К этой категории относятся задачи планирования поведения роботов и составление маршрутов передвижения транспорта.

- *Системы мониторинга* анализируют поведение контролируемой системы и, сравнивая полученные данные с критическими точками заранее составленного плана, прогнозируют вероятность достижения поставленной цели. Типовые области приложения таких систем — контроль движения воздушного транспорта и наблюдение за состоянием энергетических объектов.
- *Наладочные системы* предназначены для выработки рекомендаций по устранению неисправностей в контролируемой системе. К этому классу относятся системы, помогающие программистам в отладке программного обеспечения, и консультирующие системы.
- *Системы оказания помощи при ремонте* оборудования выполняют планирование процесса устранения неисправностей в сложных объектах, например в сетях инженерных коммуникаций.
- *Обучающие системы* проводят анализ знаний студентов по определенному предмету, отыскивают пробелы в знаниях и предлагают средства для их ликвидации.
- *Системы контроля* обеспечивают адаптивное управление поведением сложных человеко-машинных систем, прогнозируя появление возможных сбоев и планируя действия, необходимые для их предупреждения. Областью применения таких систем является управление воздушным транспортом, военными действиями и деловой активностью в сфере бизнеса.

Моделирование мышления.

GPS

В 1957 году была начата разработка системы, получившей название "Универсальный решатель задач" (General Problem Solver — GPS). Эта система создавалась коллективом из трех исследователей: Аланом Ньюэлом, Дж. Шоу и Гербертом Саймоном. Первая публикация по результатам работы появилась лишь 1959 году [Newell et al., 1959]. Проект продолжался до 1969 года.

Название "Универсальный решатель задач" обусловлено тем фактом, что это была первая система для решения задач (problem-solving system), в которой общие методы решения были отделены от знаний, определяющих конкретную задачу.

Часть программы, осуществлявшая поиск решения, не обладала информацией о конкретном виде задачи, с которой она работала.

Специфичные для конкретной задачи знания организовывались в отдельные структуры: *объекты* и *операторы* для преобразования объектов.

Задача для системы GPS ставилась в виде пары, состоящей из начального объекта и желаемого объекта, в который начальный объект должен быть преобразован.

Методология, использованная в GPS, отличается от стандартных методов поиска в пространстве состояний тем, что она выбирает путь, по которому продолжить поиск. Т.е. система пытается искать решение в первую очередь в "наиболее перспективных" ветвях поиска. Такая методология была названа "*анализ средств и целей*" (means-ends analysis).

Ее суть заключалась в том, что сначала отыскивалось *различие* (difference) между текущим объектом и объектом, который мы хотим получить. Это различие относилось к одному из ряда *классов различий*. С каждым классом был сопоставлен набор действий, способных уменьшить различие между текущим и целевым объектами.

На каждом шаге поиска GPS искал различие объектов и выбирал один из релевантных операторов, который и пытался затем применить к текущему объекту. Поиск подходящей последовательности операторов выполнялся в глубину до тех пор, пока операторы оказывались применимы, а ветвь поиска "выглядела перспективной".

Если ветвь поиска была бесперспективной, то выполнялся откат.

Важной особенностью анализа средств и целей является то, что всегда выбирается

"релевантный" оператор, уменьшающий различие объектов, даже если он и не применим к текущему объекту.

Вместо того чтобы отказаться от неприменимого к текущему объекту оператора, GPS пытался преобразовать текущий объект в объект, пригодный для применения выбранного оператора. Применение такой стратегии привело к появлению рекурсивной, целеориентированной процедуры, которая фиксирует историю поиска в графе с частично-проработанными узлами.

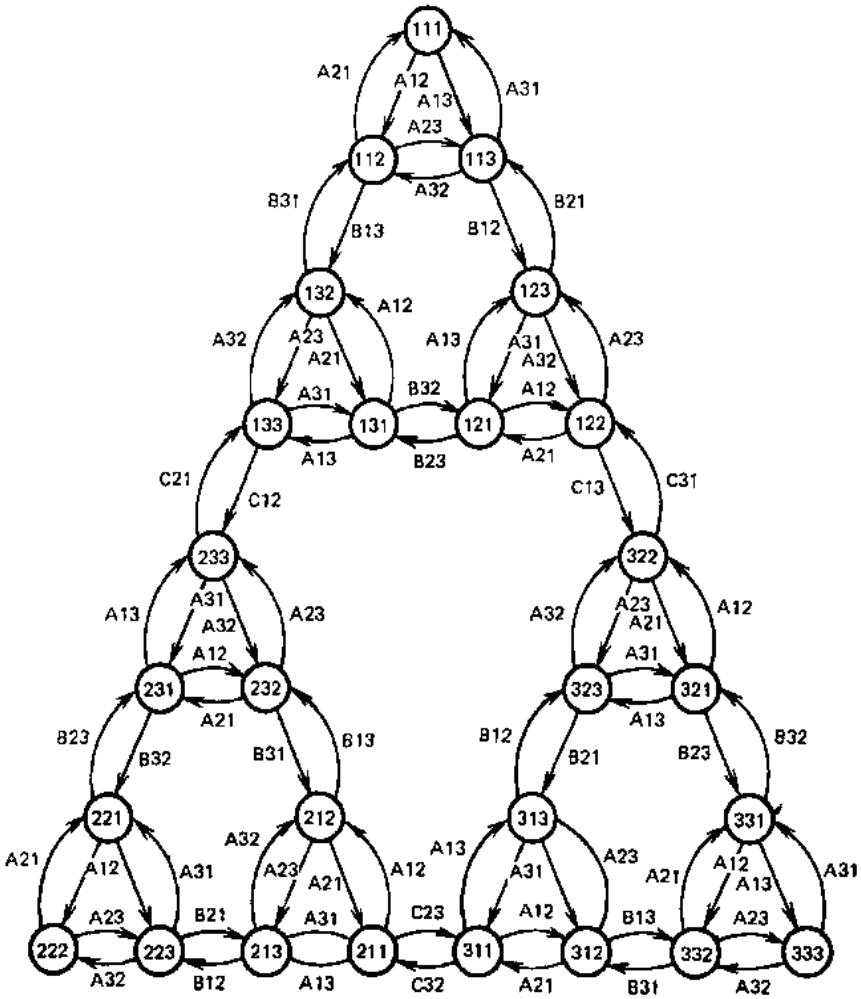


Рис. 55.

Несмотря на то, что в качестве объекта могла выступать модель мира, а в качестве оператора — действие, задача планирования перед системой GPS не ставилась. Такой вариант использования GPS был осуществлен значительно позднее, когда была разработана система STRIPS.

http://ai-center.botik.ru/planning/index.php?ptl=materials/gps-overview.htm

Планирование на базе рассуждений по прецедентам

Рассуждения по прецедентам (case-based reasoning, CBR) — это метод формирования умозаключений, опирающийся не на логический вывод от исходных посылок (логические рассуждения), а на поиск и анализ случаев формирования подобных умозаключений в прошлом. Разумеется, такие умозаключения не являются достоверными и требуют верификации. Проверка корректности умозаключения может являться частью CBR-процесса.

Системы символьных преобразований.

«Компьютерная алгебра» - на сегодняшний день это уже твердо устоявшийся термин. Область деятельности, охватываемую этим понятием, называли «аналитическими выкладками на ЭВМ», «аналитическими преобразованиями на ЭВМ», и, наконец, « компьютерной алгеброй». Широко распространено определение этого термина как «область информатики, которая занимается разработкой, анализом, реализацией и применением алгебраических алгоритмов». Наряду с этим термином также мы будем использовать термин «аналитические вычисления на ЭВМ».

Алгебраическое вычисление выражения можно рассматривать как последовательное преобразование выражения согласно некоторому множеству правил до тех пор, пока к выражению нельзя будет применить ни одного правила.

Аналитическое вычисление не сталкивается с такими проблемами численного программирования как ошибки округления, проблемы сходимости и проблемы устойчивости.

В системах компьютерной алгебры алгебраические вычисления выполняются точно.

Система компьютерной алгебры является развитой системой программирования, но в качестве базовых типов данных практически всегда рассматриваются формулы и формула является объектом преобразования. Формулы можно дифференцировать и интегрировать, в формулы можно подставлять вместо переменных новые формулы и выполнять много различных преобразований.

Существует достаточное количество систем компьютерной алгебры, которые с легкостью выполняют трудоемкие алгебраические преобразования. К ним относятся такие, как Reduse, Maple, Mathematica.

Maple - система символьных вычислений занимает в этой отрасли наряду с системой *Mathematica* фирмы Wolfram Research ведущие позиции. Она была создана канадской группой символьных вычислений.

В Maple имеется компактное ядро, написанное на языке Си, которое реализует интерпретируемый язык, удобный для написания алгоритмов компьютерной алгебры.

Основная часть системы представляет знания, которые содержатся в библиотеке, написанной на этом языке.

Система обладает громадным (свыше 2500) набором самых различных функций для выполнения аналитических и численных вычислений, решения алгебраических и дифференциальных уравнений, графического вывода результатов и многих других действий. *Maple* работает на широком круге платформ.

Priz

http://www.ershov.ras.ru/archive/eaimage.asp?lang=1&did=8230&fileid=103548

"МИКРОПРИЗ - система программирования для ПЭВМ".

Решатель

Основная задача решателя — это преобразование исходного описания задачи или задания на выполнение некоторой программы в рабочую программу для ЭВМ.

Математически эта задача сводится к переводу исходного описания в некоторое промежуточное формализованное описание (описание на языке спецификаций, в качестве которого чаще всего выступает исчисление предикатов первого порядка или некоторое подмножество формул, допустимых в подобном исчислении).

Подобная формализация рассматривается как теорема. Доказательство теоремы можно найти в рамках формальной системы, все компоненты которой хранятся в базе знаний.

Средства вывода входят в решатель. Построенный вывод (при положительном результате доказательства) позволяет извлечь из него рабочую программу, нужную

пользователю.

Работы по реализации подобных процедур были начаты в СССР довольно давно, раньше, чем в зарубежных странах.

Первые системы такого типа ПРИЗ и СПОРА заложили основу теории и практики построения подобных систем.

Для решения задач в ПРИЗ использовался специальный язык – УТОПИСТ. С его помощью на естественном для человека языке формулировалась задача. Например, для закона Ома U = R * I задача могла быть сформулирована так: «По известным сопротивлению и току определить напряжение», или так: «по известным напряжению и току определить сопротивление». Во втором случае формулу нужно изменить: R = U / I.

ПРИЗ символьных преобразований не выполнял. Его решатель содержал большое количество программных модулей, каждый из которых имел стандартное описание: сначала перечислялись исходные данные, а затем – результирующие. Планировщик по постановке задачи анализировал, что известно, и что надо определить, а затем подбирал программные модули так, чтобы получить необходимые данные по известным исходным величинам.

Позднее выяснилось, что в качестве языка, на котором функционирует решатель, можно использовать языки логического программирования, в частности язык ПРОЛОГ. Переход к языкам такого типа резко повышает эффективность процедур автоматического синтеза программ в решателях.

Средствами системы программирования высокого уровня ПРИЗ, которую ее идеолог Э.Х. Тыугу квалифицирует как инструмент концептуального программирования, реализована, СУБД DABU.

АРИЗ – алгоритм решения изобретательских задач.

www.altshuller.ru

Алгоритм решения изобретательских задач (АРИЗ) - комплексная методика алгоритмического типа, основанная на законах развития технических систем и предназначенная для анализа и решения изобретательских задач.

АРИЗ возник и развивался вместе с Теорией Решения Изобретательских Задач (ТРИЗ). Первоначально АРИЗ назывался "методикой изобретательского творчества".

Впервые словосочетание "алгоритм решения изобретательских задач" использовано в приложении "Технико-экономические знания" к еженедельнику "Экономическая газета" за 1 сентября 1965 г.

Аббревиатура АРИЗ впервые использована в книге Г.С. Альтшуллера "Алгоритм изобретения", Московский рабочий, 1-е изд.: 1969, 2-е изд.: 1973.

В дальнейшем модификации АРИЗ включали указание на год публикации, например АРИЗ-68, АРИЗ-71, АРИЗ-77, АРИЗ-82, АРИЗ-85.

Автор АРИЗ – Генрих Саулович Альтшуллер.

В 70х годах в нескольких исследовательских центрах ТРИЗ началась работа над программным обеспечением, предназначенным для автоматизации решения изобретательских задач. Этот проект получил название "Изобретающая машина" (ИМ).

С падением железного занавеса и развалом СССР ТРИЗ (как и все прочие жизнеспособные отрасли советской науки) активно "вывозился" на Запад, в немалой степени за счет утечки мозгов.

В течение последнего десятилетия центр развития АРИЗ переместился в США, где сейчас находятся две лидирующие компании, производящие программное обеспечение на его базе: Invention Machine и Ideation.

Программа TechOptimizer сегодня признается одной из наиболее совершенных в мире ИМ. В качестве лаконичного, но красноречивого подтверждения этого факта

достаточно привести несколько пунктов из длинного списка организаций, использующих TechOptimizer:

NASA, BMW, Boeing, Intel, General Electric, Shell и т. д.

Среди коммерческих разработок, продолжающих проводиться в области ИМ на постсоветском пространстве, выделяется программа IdeaFinder, созданная московским специалистом по ТРИЗ А. Барышниковым - "изобретающая машина" с русскоязычным интерфейсом.

АРИЗ имеет иерархическую структуру.

Основная цель первой части АРИЗ - переход от расплывчатой изобретательской ситуации к четко построенной и предельно простой схеме (модели) задачи.

Анализ по первой части АРИЗ и построение модели существенно проясняют задачу и во многих случаях позволяют увидеть стандартные черты в нестандартных задачах.

Это открывает возможность более эффективного использования **Ошибка! Недопустимый объект гиперссылки.**, чем при применении их в исходной формулировке задачи.

Цель второй части АРИЗ - учет имеющихся ресурсов, которые можно использовать при решении задачи: ресурсов пространств, времени, веществ и полей.

В результате применения третьей части АРИЗ формулируется образ идеального решения (ИКР). Определяется также и физическое противоречие (ФП), мешающее достижению ИКР. Не всегда возможно достичь идеального решения. Но ИКР указывает направление на наиболее сильный ответ.

Четвертая часть АРИЗ включает планомерные операции по увеличению ресурсов: рассматриваются производные ВПР (вещественно-полевые ресурсы), получаемые почти бесплатно путем минимальных изменений имеющихся ВПР.

Во многих случаях четвертая часть АРИЗ приводит к решению задачи. В таких случаях можно переходить к **Ошибка! Недопустимый объект гиперссылки.**. Если же после Ошибка! Недопустимый объект гиперссылки. части ответа нет, надо пройти пятую часть.

Цель пятой части АРИЗ - использование опыта, сконцентрированного в информационном фонде ТРИЗ. К моменту ввода в пятую часть АРИЗ задача существенно проясняется - становится возможным ее прямое решение с помощью информационного фонда.

Простые задачи решаются буквальным преодолением физического противоречия, например разделением противоречивых свойств во времени или в пространстве.

Решение сложных задач обычно связано с изменением смысла задачи - снятием первоначальных ограничений, психологической инерцией и до решения кажущихся самоочевидными.

Например, увеличение скорости "ледокола" достигается переходом к "ледоНЕколу".

Для правильного понимания задачи необходимо ее сначала решить: изобретательские задачи не могут быть сразу поставлены точно. Процесс решения, в сущности, есть процесс корректировки задачи.

На шаге 6 если задача решена, формулируется способ и принципиальная схема устройства, осуществляющего этот способ.

Если ответа нет, проверяется - не является ли **Ошибка! Недопустимый объект гиперссылки.** сочетанием нескольких разных задач. Если “да”, то следует изменить **Ошибка! Недопустимый объект гиперссылки.**, выделив отдельные задачи для поочередного решения (обычно достаточно решить одну главную задачу).

Главная цель седьмой части АРИЗ - проверка качества полученного ответа. Физическое противоречие должно быть устранено почти идеально, "без ничего". Лучше потратить 2-3 часа на получение нового - более сильного - ответа, чем потом полжизни бороться за плохо внедряемую слабую идею.

Восьмая часть АРИЗ имеет целью максимальное использование ресурсов найденной идеи. Действительно хорошая идея не только решает конкретную задачу, но и дает универсальный ключ ко многим другим аналогичным задачам.

Каждая решенная по АРИЗ задача должна повышать творческий потенциал человека. Но для этого необходимо тщательно проанализировать ход решения. В этом смысл девятой (завершающей) части АРИЗ.

Наиболее влиятельной организацией, занимающейся ТРИЗ, является в настоящее время Международный институт Альтшуллера, а среди печатных изданий можно отметить специализированный ежемесячник "Журнал ТРИЗ". На постсоветском пространстве ТРИЗ более всего развит в Минске. Местный исследовательский центр организует преподавание элементов ТРИЗ в школах и даже посвящает им специальные еженедельные телепрограммы.

Экстенсивное развитие теории ТРИЗ, изначально ориентированной на технические проблемы, успешно адаптирует ее к решению задач

- психологии,
- экономики
- и программирования.

Например, по адресу www.trizjournal.com/archives/2001/05/a/index.htm находится статья, рассказывающая **о применении ТРИЗ для программирования на языке Perl**, а www.trizland.ru служит наглядным примером использования ТРИЗ для создания развлекательного сайта.

Рис. 56.

Работа над заголовком презентации, Web-страницы. (Программа headliner).

На специализированном сайте http://headliner.triz-ri.ru содержится программа, облегчающая конструирование заголовков. Она сопровождается следующим описанием:

«Неоднократные очные и заочные опросы журналистов, рекламистов, PR-профи, проведенные консалтинговой фирмой "ТРИЗ-ШАНС", выявили три проблемы, наиболее трудные для современного "человека пишущего", независимо от того, что он пишет - статью, сценарий или текст речи:

- Как увлекательно начать материал?
- Чем и как закончить, чтобы это запомнилось аудитории?
- Как придумать заголовок, слоган, метафору?

Для решения третьей задачи была создана программа "HeadLiner/Заголовщик".

Человек, далекий от журналистики и PR, не воспринимает создание заголовка как сложную задачу. Однако когда этим приходится заниматься постоянно, особенно в условиях дефицита времени, то эффективнее работать с помощью профессионального инструмента, чем без него.

Итак, программа "HeadLiner/Заголовщик" разработана с целью повысить эффективность создания каждым Автором:

- заголовков;
- рекламных девизов, слоганов, лозунгов, эхо-фраз;
- образных фраз, метафор;
- текстов для баннеров;
- поиска нужных цитат (например, высказываний политических деятелей).

Учитывая, что слоганы и заголовки часто являются видоизменением известных выражений, программа содержит 10 тематических баз данных с возможностью их просмотра при работе:

- изречения из Ветхого и Нового завета;
- пословицы и поговорки русского языка;
- фразы из мульт- и кинофильмов;
- "перлы" российских политиков XIX-XX веков;
- современный жаргон;
- афоризмы Козьмы Пруткова, Ежи Леца и др.

Демо-версию программы можно найти по адресу: http://www.altshuller.ru.

Программа headliner:

Сначала необходимо поставить задачу: Пользователь выбирает параметры, характеризующие его ситуацию, и через несколько секунд получает подборку решений-аналогов (от нескольких штук до нескольких сотен - в зависимости от настроек программы) для последующих размышлений.

Работу с программой "HeadLiner/Заголовщик" можно кратко охарактеризовать с помощью трех слов:

ИГРА - АНАЛОГИИ - ЦИКЛ

Почему так?

ИГРА - потому что рекомендуется относиться к программе без излишней серьезности, как к игре с интеллектуальным партнером, который постоянно подкидывает Вам варианты решений.

АНАЛОГИИ - ибо программа сама не создает фраз - она лишь пробуждает с помощью аналогий интуицию Пользователя. И Пользователь благодаря этим аналогиям создает заголовки, фразы, метафоры.

ЦИКЛ - так как если не все получилось с первого раза - необходимо просто сделать еще один цикл работы с программой.

В программе всего два основных рабочих экрана.

ЭКРАН ПОСТАНОВКИ ЗАДАЧИ

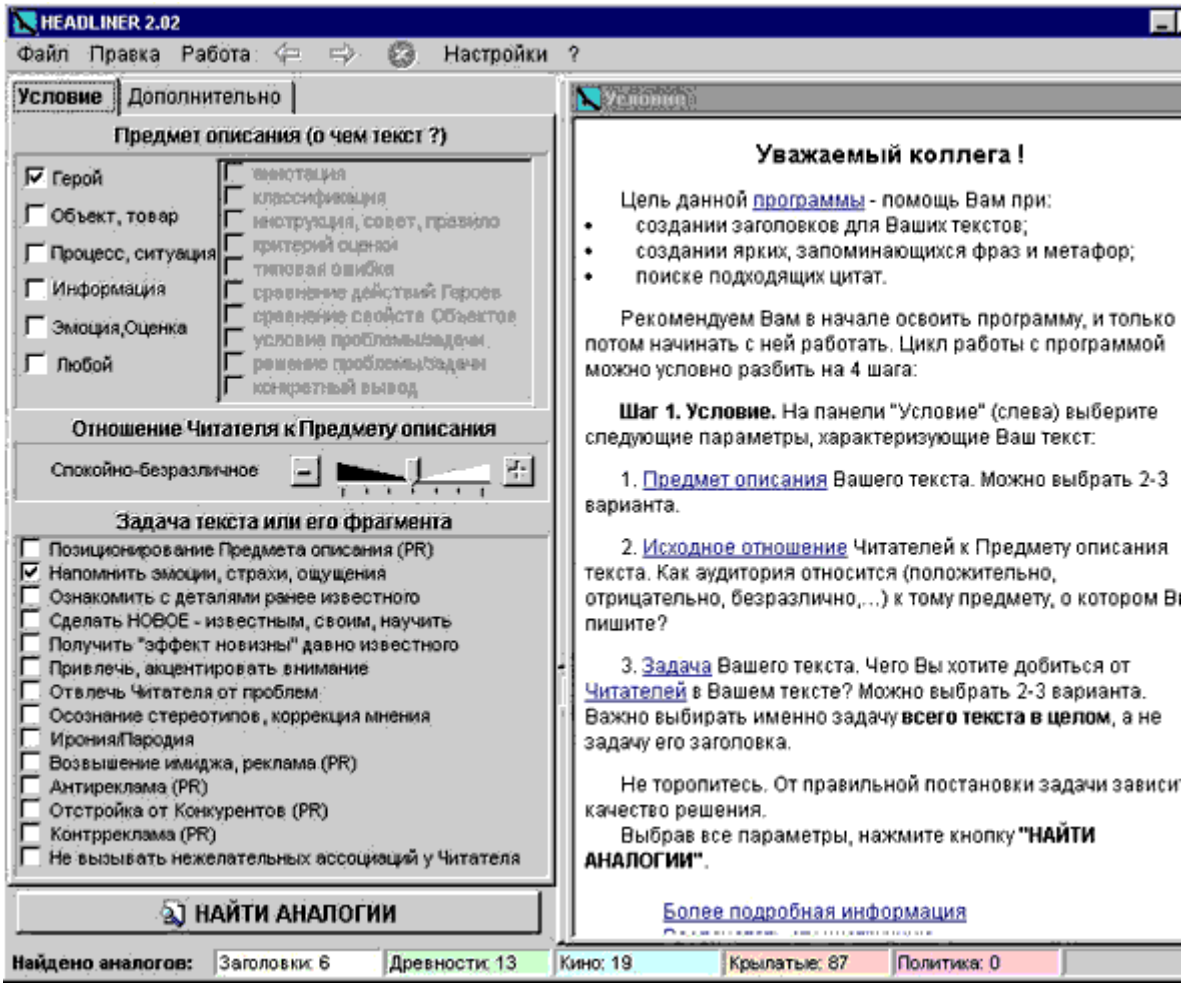


Рис. 57. Основной экран HeadLiner.

и **ЭКРАН РЕШЕНИЙ-АНАЛОГИЙ,** на котором отображаются выдаваемые программой решения

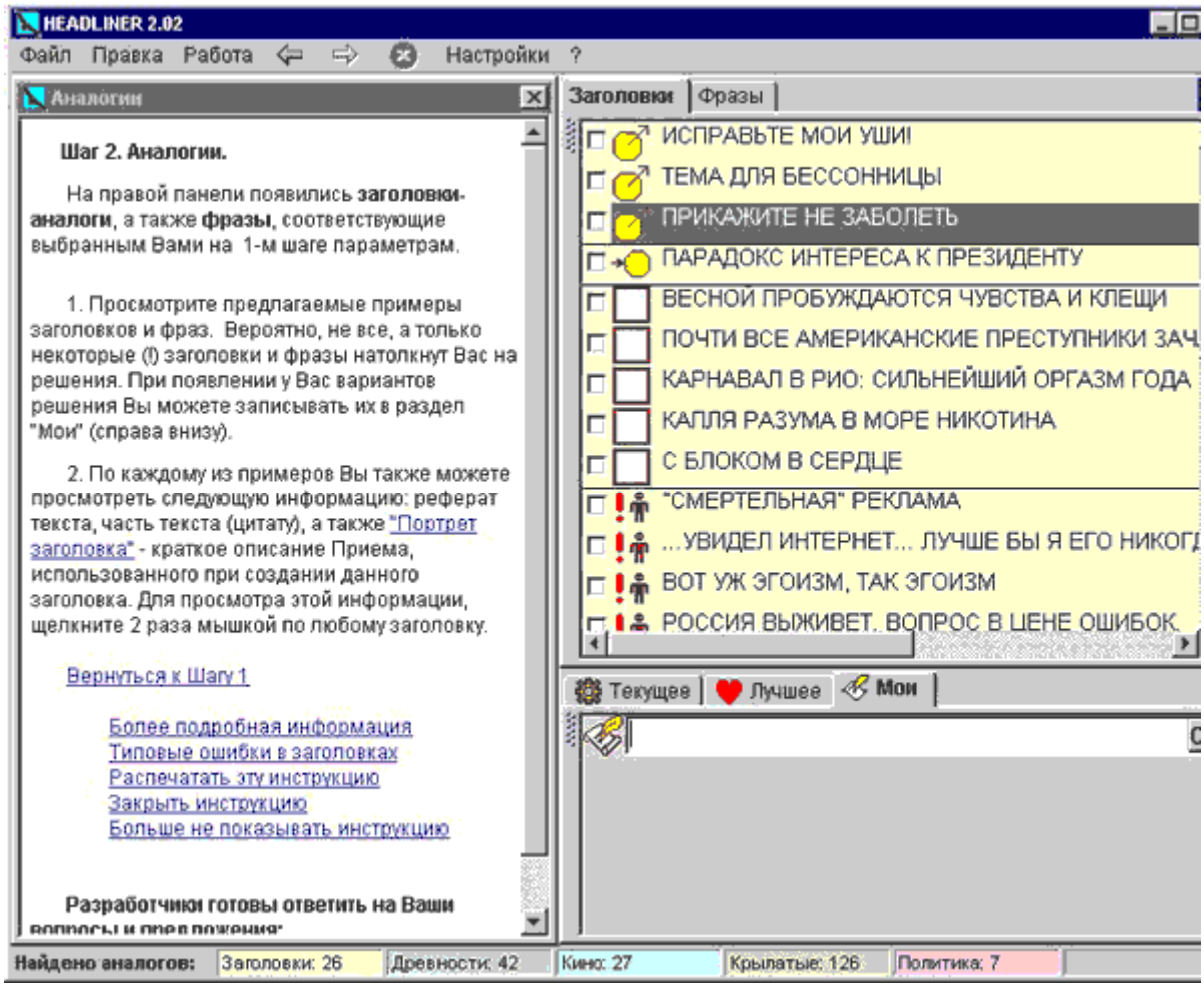


Рис. 58. Экран решений – аналогий.

Каждый заголовок-аналог, попавший в базу программы, снабжен рефератом и фрагментом текста, к которому он относится, а также "Портретом заголовка" - описанием его структуры, "формулой" решения. В "Портрет заголовка" входит раздел "Содержание (тема) текста", помогающий Пользователю отследить связь заголовка с основными ресурсами цитируемого текста.

После постановки задачи возможны два режима работы с программой. Пользователь может использовать тот режим, который ближе его задаче и его стилю.

РЕЖИМ 1

МЕТОД ПРЯМОЙ АНАЛОГИИ

Пользователь создает заголовок (слоган, метафору) путем прямой аналогии с предлагаемыми программой примерами. Возникающие по ходу работы варианты можно записывать в папку "Мои".

РЕЖИМ 2

МЕТОД СИНТЕЗА

Пользователь отбирает понравившиеся ему варианты, нажимает клавишу "СИНТЕЗ" и ...получает формулу (прием) создания подобных заголовков. Причем программа позволяет построить три различных формулы решения:

- стандартный вариант (например, для газеты "Труд");
- оригинальный вариант (например, для журнала "Огонек");
- экзотического вариант (например для авангардного издания).

Первые два варианта могут пригодиться при работе с массовой, а последний - при работе с узко-авангардной аудиторией...

В конечном счете, то, каким инструментом воспользоваться и каким будет искомое решение, выбирает уважаемый Пользователь - программа ему только помогает. Программа призвана усиливать - но не заменять (!) - интуицию, знания и образы человека. Именно поэтому у разных Авторов, использующих программу, результаты получаются разными...

Разработчики рекомендуют ознакомиться с системой Help программы (для чего достаточно нажать клавишу F1), содержащей, в том числе:

- описание типовых ошибок Авторов при создании заголовков;
- рекомендации, как эмоционально усиливать заголовки;
- указания, при каких условиях разумно переходить от отдельных заголовков к их системе.

Вот пример работы с программой HeadLiner. Исходную задачу поставил Сергей Михайлов на Internet-форуме http://www.triz-ri.ru: "Помогите придумать завлекающее название для книги о скорочтении и памяти".

На панели "Постановка задачи" мы выбрали следующие параметры, на наш взгляд, довольно точно характеризующие данную ситуацию:

Предмет описания: Услуга; Информация; Эмоция.

Исходное отношение Читателей: скорее всего - спокойное.

Задачи, решаемые книгой: Научить Читателей новой для них методике; Получить эффект новизны.

Из выданных программой примеров были отобраны несколько десятков, вот некоторые из них:

ИЗОБРЕСТИ СПОСОБ ИЗОБРЕТАТЬ
СТАТЬ В ЖИЗНИ МАСТЕРОМ
ПО ЗАКОНАМ ПРИРОДЫ ЭТО НЕ ДОЛЖНО ЛЕТАТЬ
ФИЗКУЛЬТУРА С ПЕЛЕНОК И... ДАЖЕ РАНЬШЕ
ЗРЕНИЕ ТРЕМЯ ГЛАЗАМИ
ВЗГЛЯД ИЗ БУДУЩЕГО ВСЕГДА СПОКОЕН
РАЗВИТИЕ ТЕХНИКИ - ВЗГЛЯД С ПТИЧЬЕГО ПОЛЕТА
ГОРЯЧИЙ ЧАЙНИК В РОЛИ УЧИТЕЛЯ
ВОСПЕТЬ ИЛИ ПРОКЛЯСТЬ?
ПОЙМАННОЕ ПРОСТРАНСТВО
ВЫШКА НА СЕКУНДУ
НОВОСТИ МЫШЕСТРОЕНИЯ
МЕТОД ПРОБ. БЕЗ ОШИБОК
ПРАВИЛА ХОРОШЕГО ТОНА
СДЕЛАТЬ ПРОЩЕ, ЧЕМ ПРОЧЕСТЬ
СТУЧИТЕ, И ОТВОРЯТ ВАМ и др.

После получаса размышлений над этими и другими примерами, а также рекомендациями программы - были получены следующие "заготовки" названия:

ПРИРУЧИТЕ СВОЮ ПАМЯТЬ!
ВАШИ СПОСОБНОСТИ - МОСТИК В БУДУЩЕЕ
СПОСОБНОСТИ: ПРИРОДА ИЛИ ВОСПИТАНИЕ?
СПОСОБНОСТИ ПО ПЛАНУ
НОВОСТИ ПАМЯТЕСТРОЕНИЯ
ПРАВИЛА ХОРОШЕЙ ПАМЯТИ
КАК ПРОЧИТАТЬ ЭТУ КНИГУ ЗА 5 МИНУТ
20 КНИГ В ДЕНЬ - ЭТО НОРМАЛЬНО
МЫ НАУЧИМ ВАС ГЛОТАТЬ КНИГИ
КНИГА О ТОМ, КАК ГЛОТАТЬ КНИГИ

"Заготовки" для подзаголовков книги, а также для ее рекламы:
В ОБХОД СОПРОТИВЛЕНИЯ
ВДОХНОВЕНИЕ - БИБЛЕЙСКОЕ СЛОВО
ПУТЬ СИЛЬНЕЕ ЧЕЛОВЕКА
СЛАБОВОЛИЕ КАК ПРЕДРАССУДОК
ОТКРЫТИЕ ВОПРЕКИ СЕБЕ
ЧТО ОБЫЧНО НЕДОГОВАРИВАЮТ
ЧЕМ УДИВЛЯТЬ БУДЕМ?
ВРЕМЯ МЕНЯТЬ ПАМЯТЬ...
САМОУЧКИ ТЕРЯЮТ ВРЕМЯ
ПОЙМАННАЯ ПАМЯТЬ

Итого - 20 неплохих вариантов за полчаса. Есть из чего выбрать...

Заказать профессиональную версию программы "HeadLiner/Заголовщик" можно по e-mail разработчиков: PR1@online.ru».

Мышление.

Структурные единицы знаний.

Структурными единицами знаний являются

- понятия,
- доказательные и правдоподобные рассуждения.
    - Доказательные рассуждения – это теоремы.
    - Правдоподобные рассуждения – это интуитивно истинные или истинные по жизненному опыту рассуждения.
- утверждения
- стереотипные процедуры.

Стереотипы обычно относят к актуальным знаниям, всё остальное – к фактуальным.

Эти структурные единицы Мински в 1974г. назвал фреймами.

Примерами фреймов являются:

- Понятие «треугольник»;
- Доказательное утверждение о равенстве треугольников по двум сторонам и углу между ними;
- Правдоподобное рассуждение: «дети наследуют характер родителей»;
- Стареотипная процедура заказа авиабилетов.

Среди важнейших классов задач, которые ставились перед *ИИ* с момента его зарождения как научного направления (с середины 50-х годов XX века), следует выделить следующие трудно формализуемые задачи:

- *доказательство теорем*,
- *управление роботами*,
- *распознавание изображений*,
- *машинный перевод* ,
- *понимание текстов на естественном языке*,
- *игровые программы*,
- *машинное творчество* (*синтез музыки*, *стихотворений*, текстов).

Псевдомыслительные алгоритмы.

Самый простоя “перифраз” позволяет создать и поддерживать иллюзию понимания. Это видно из опытов, проводимых Вейценбаумом с программой “Элиза”. Фрагмент беседы с этой программой выглядит, как будто разговор ведут понимающие друг друга собеседники:

Человек: “Вы не очень настойчивы, но я знаю, что Вы не хотите, чтобы я это заметил”;

ЭВМ: “Почему Вы думаете, что я не очень настойчива?”

Человек: “Вы со мной не спорите”;

ЭВМ: “Почему Вы думаете, что я с Вами не спорю?”

Человек: “Вы меня боитесь”;

ЭВМ: “Вам доставляет удовольствие считать, что я Вас боюсь?”...

Эта и последующие программы Вейценбаума нашли применение для борьбы с одиночеством, очень трудно переживаемым людьми, находящимися среди тех, кто их “не понимает”. Бездушная, ничего не понимающая программа, работающая на принципе перифраза и стандартных заготовок, способна “облегчить душу” одинокому человеку, снять накопившееся у него эмоциональное напряжение. Это еще раз подчеркивает, что человеку необходимо общение - самое страшное для него - это одиночество.

В [Кондрашов “Электронный разум”, М., Знание, 1987.] приводится фрагмент беседы программы “Доктор (Д)” с больным (Б) того же автора:

Д: “Здравствуйте, я - Доктор. Садитесь, пожалуйста, за пишущую машинку и расскажите мне, что Вас беспокоит”.

Б: “Я хотел бы, чтобы у меня был мой багаж”.

Д: “Вы можете уточнить свой ответ?”

Б: “Я не совсем ясно понимаю, какого рода уточнение требуется”.

Д: “Вы полагаете, что это нормально, не совсем ясно понимать, какое уточнение требуется?”

Б: “Да, совершенно нормально”.

Д: “Понимаю”. (Для того, чтобы погасить возникший эмоциональный накал, программа применяет “пропуск транзакции”, отдает инициативу больному).

Б: “Я потерял свой багаж”.

Д: “Вы говорите, что потеряли свой багаж?” (программа задает нейтральный вопрос, инициативу не берет).

Б: “Я чувствую себя без него потерянным”.

Д: “И часто Вы чувствуете себя потерянным без него?”

Б: “Нет, это - впервые”,

Д: “Почему?”.

Б: “Потому, что обычно я не теряю своих вещей”.

Д: “Ваши родители теряли Вас когда - нибудь?”...

Заключенный в скобки комментарий показывает, что при общении эмоциональным фоном можно управлять. Но для этого хотя бы один из участников должен осознавать цель, знать возможности - свои и партнера. Этот участник является инициатором проводимых операций.

Системы человеко-машинного общения.

Системы человеко-машинного общения должны создавать положительный эмоциональный фон, который возникает, если в процессе общения признаются достоинства человека, подтверждается эмоционально-психологическая стабильность, сохранение достигнутого равновесия, душевного комфорта, внутренней гармонии (по-японски - “ва”).

Программисты выработали три шутливых правила, выполнение которых способствует достижению этой цели:

1) Ничего не ожидайте от пользователя. Считайте, что он все будет делать неправильно;

2) Уважайте своих пользователей и защищайте их;

3) Сделайте так, чтобы они не страдали от собственной некомпетентности.

Структурная схема интеллектуальной системы человеко-машинного общения имеет вид:

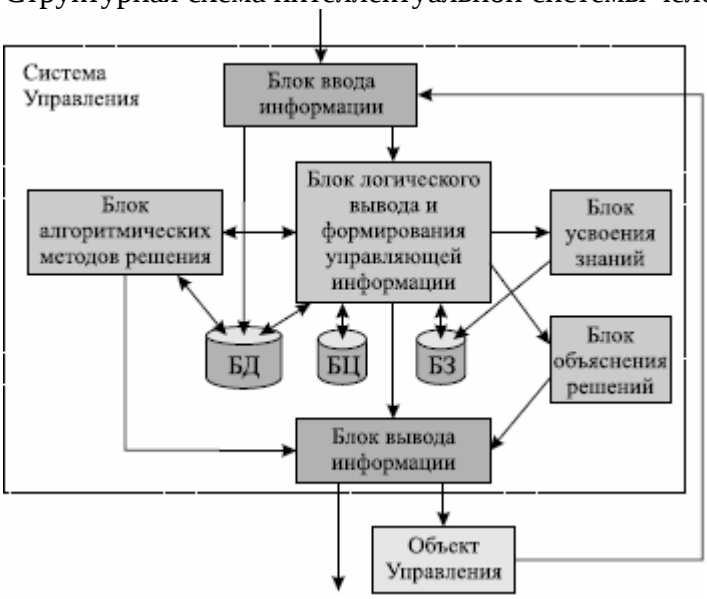


Рис. 59.

Система речевого общения.

Несмотря на бурное развитие технических средств общения, в том числе - для обработки звука на ЭВМ, широкого распространения речевое общение с ЭВМ не получило. Причина этого заключается в низком качестве человеко-машинных интерфейсов. Качество их оставляет желать лучшего, так как (на примере речевого интерфейса):

Во-первых, они пассивны: они работают на заднем плане, пытаясь угадать, что же именно им сказали. А системы распознавания речи обладают хотя и достаточно высокими, но все-таки ограниченными возможностями (даже если 90% раздельно произнесенных слов распознается правильно, то остается 10%, распознаваемых неверно. А если эта система воспринимает слова, как команды, то 10% ее действий будет носить диверсионный характер, например, стирать файлы, форматировать диски с очень важной информацией, передавать одному абоненту сообщение, предназначенное для другого, и т.д.).

Первый вывод из этого (лежащий на поверхности) - надо совершенствовать программы распознавания, доводя их надежность до 100%. Но ведь человек при общении распознает далеко не 100% слов, и это не приводит к катастрофическим последствиям. Это наводит на мысль, что надо совершенствовать не программы распознавания (для которых может быть достаточно обеспечивать 70-80% распознавание речи), а речевой человеко-машинный интерфейс: он должен быть активным, уметь перехватывать инициативу - переспрашивать пользователя при сомнениях, подтверждать получение команды, всегда быть готовым к отмене неверно распознанной команды, и др.

Во время общения даже при отсутствии сомнений в правильности распознавания, ЭВМ не должна сохранять молчание (попробуйте пообщаться по телефону с человеком, который все время молчит - может создаться впечатление, что он Вас игнорирует). Как минимум, пользователю необходимо услышать “Да-а”, или “Ага”, чтобы быть уверенным, что его слушают, воспринимают, с ним соглашаются,... Необходима обратная связь, которая при человеко-машинном общении может проявляться во вставке машиной не несущих информации реплик типа: “Конечно”, “Хорошо”, “Понимаю”, а при

возникновении сомнений: “Простите?”, “Вы сказали...?” и др. Такие реплики сделают интерфейс более дружественным и позволят компенсировать недостаточную достоверность систем распознавания речи.

Во-вторых, речевые человеко-машинные интерфейсы не учитывают психологических особенностей общения: они могут быть назойливыми, бестактными, глупыми, неуместными. Например, в одном из японских автомобилей была реализована система оповещения водителя о неплотном закрывании дверей. Эта система включалась и монотонно повторяла: “Дверь открыта, дверь открыта,...” и отключить ее было невозможно. Очень скоро тупость этой системы начинала раздражать водителя и пассажиров, и они были счастливы, если им удавалось сломать систему и выключить (навсегда) надоедливый голос.

Реализованная в Белоруссии система речевого оповещения клиентов о наступлении срока уплаты за телефон не имела обратной связи - клиент мог только выслушать сообщение, и был лишен возможности возразить на несправедливый упрек. Естественно, такая система долго не просуществовала, эксперимент был признан неудачным и из-за полученного “отрицательного результата” финансирование работ по разработке речевых систем человеко-машинного общения было прекращено.

Транзакции речевого общения должны соответствовать общепринятому ритуалу. Например, правила хорошего тона определяют, что не следует заговаривать с кем-либо, пока к Вам не обратились, на Вас не обратили внимание, тем более, если лицо, к которому Вы хотите обратиться, занято. Если машина хочет сообщить оператору нечто очень важное, она не должна начинать сообщение без приглашения - она должна вежливо произнести “Гхм”, или “Простите” и ожидать ответа. Когда оператор сочтет возможным и скажет: “Да”, или “Слушаю”, или что-нибудь подобное, ЭВМ может выдать накопленную информацию.

Общение с ЭВМ должно быть естественным для человека! Если это не соблюдается, вместо речевого общения будут использоваться другие средства - клавиатура, мышь, сенсорный дисплей и др.

Естественность проявляется не только в вежливости. Наряду с вежливостью могут использоваться и разговорный язык, и произношение с акцентом, и юмор - компьютер должен генерировать бодрость, готовность помочь, жизнерадостность. Например, вместо сухого “Хорошо”, ЭВМ может произнести: “Бусделно”, или “Нет проблем”, или “Все, что пожелаете”, и т.д.

ЭВМ может воспринять ограниченное количество слов, но произнести она может гораздо больше - это дает возможность варьировать ответы, чтобы избежать монотонного общения.

В третьих, речевые человеко-машинные интерфейсы не учитывают, что разборчивость речи сильно зависит от используемых программно-технических средств. Встроенный динамик может произносить речевые сообщения недостаточно громко и неразборчиво. А если качество записи эталонных фраз было низким, то дикция ЭВМ будет соответствующей. И проблема распознавания речи будет возникать не у компьютера, а у человека - оператора! Речевой интерфейс должен это учитывать и воспринимать реплики оператора, распознавать их и реагировать на них, корректируя выводимое сообщение. Другими словами, система общения должна работать в дуплексном режиме.

В четвертых, речевые интерфейсы должны учитывать возможную витиеватость речи оператора, образность выражений, использование синонимов, и др. Это предъявляет повышенные требования к возможностям ЭВМ по выявлению смысла сообщения. Для снижения этих требований естественноязыковое общение можно реализовать в виде “деловой прозы” с использованием перифраза для уточнения смысла сообщения. Но самое простое для реализации - это использование оператором языка команд со “свободным ответом” компьютера.

В пятых, возможности речевого интерфейса должны быть понятны оператору. Чтобы обеспечить оператора информацией о возможностях данного интерфейса, в состав его должна входить обучающая система. Обычно система обучения в речевом интерфейсе используется для ознакомления ЭВМ с особенностями речи конкретного оператора, настройки на оператора, и **НЕ** содержит информации для оператора (или содержит очень краткие сведения) о возможностях интерфейса. Это затрудняет “срастание” человека с данным интерфейсом, привыкание их друг к другу. В интерфейсе должна присутствовать обучающая система, дополняющая систему оперативной помощи, отвечающей на вопросы типа “Как?” и “Почему?”.

При разработке речевого интерфейса необходимо начинать со сценария общения. Необходимо помнить, что общение человека с ЭВМ - это не общение с бездушной железкой, а общение двух людей - того, кто создал интерфейс с тем, кто им пользуется. Поэтому сценарий общения должен разрабатываться, исходя из того, как бы выглядело общение в разговоре с живым человеком.

Когда сценарий закончен, надо, учитывая технические ограничения, сделать нечто, максимально похожее на живую речь.

Набор команд, воспринимаемых машиной, должен быть простым, компактным, минимально подверженным ошибкам, и соответствующим лексике пользователя. Речевые ответы должны быть естественными, соответствующими лексике пользователя, вежливыми, бодрыми, жизнерадостными. Можно предположить, что такой речевой интерфейс быстро приживется и будет активно использоваться пользователями.

Проблема понимания.

В основе общения лежит проблема понимания. Можно знать наизусть всего «Евгения Онегина», и не ответить на вопрос: какое отчество было у Татьяны Лариной, или на вопрос: когда состоялась дуэль Ленского с Онегиным.

Лингвистическая футурология изучает грядущее, исходя из трансформационных возможностей языка: ***человек в состоянии овладеть только тем, что может понять,*** *а понять он может только то, что выражено словами. Невыраженное словами ему недоступно.*
С. Лем. Футурологический конгресс

Д.А.Поспелов сформулировал 5 уровней понимания:

1) прямой анализ текста, без привлечения каких - либо дополнительных знаний;

2) привлечение информации из памяти - чаще всего “бытовых знаний” о временной, пространственной и причинно-следственной структуре анализируемого текста;

3) использование всей информации по теме, имеющейся в памяти, полученной при прямом анализе текста, включая знания, отраженные на втором уровне понимания;

4) привлечение ассоциаций и аналогий - тех фрагментов памяти, которые напрямую могут быть и не связаны с анализируемым текстом, но “близки” ему;

5) из текста извлекается его прагматическое содержание за счет сопоставления выясненного на предыдущих четырех уровнях смысла со своими целевыми установками, со своей системой ценностей, со своими возможностями, ресурсами. При этом учитывается прогноз развития событий, производится планирование своих действий для достижения основной цели.

Смысл текста не является совокупностью смыслов образующих его предложений. Понимание смысла связано с выполнением умозаключений, с использованием интеллектуальных навыков, включающих в себя такие, как:

- сопоставление сложных объектов и оценку их сходства;
- выделение типового объекта из группы однородных;

- поиск типичных черт, существенных признаков;
- формирование описания типового объекта, выделение его отличительных черт;
- определение понятий (дефиниции);
- выявление причинно-следственных связей;
- интерпретация связей и свойств исследуемых объектов;
- генерация гипотез;
- выявление закономерностей;
- самообучение, адаптация;
- умение делать традуктивные, индуктивные, дедуктивные выводы;
- ...

Даже на уровне прямого анализа текста это не простая задача: каждое предложение может нести в себе лингвистически невыраженный подтекст, выявление которого является задачей пресуппозиций.

Очень ярко это показано в [Э.В.Попов «Общение с ЭВМ на естественном языке» М., Наука, 1982], где приводятся следующие пресуппозиции для фразы ”Врач бегло говорила по-немецки”:

- врачем является женщина;
- немецкий для врача - не родной язык;
- отмечается степень владения языком, а не процесс говорения;
- дается оценка культурного уровня врача;
- умение “говорить по-немецки” относится к прошлому;
- имеется в виду определенный человек.

Генетические алгоритмы (ГА),

Изобретены в начале 60х гг., в основном предназначены для решения оптимизационных задач.

Структуры данных, которыми они манипулируют, являются метафорами живых организмов, использующих механизмы скрещивания, мутационной изменчивости и естественного отбора для приспособления к условиям окружающей среды.

Генетическое программирование (ГП) применяет указанную метафору не только к данным, но и к программному коду. Эта область, считавшаяся бесперспективной вплоть до начала 90х годов, теперь бурно развивается. Признанным лидером в ней является Джон Коза (John Koza) профессор Стэнфордского университета и основатель компании Genetic Programming.

Типовой задачей для построения генетических алгоритмов является задача “Умный муравей”.

Краткое описание задачи “Умный муравей”.

Используется двумерный тор размером 32 на 32 клетки. На некоторых клетках поля расположены яблоки – черные клетки на рис. Яблоки расположены вдоль некоторой ломаной линии, но не на всех ее клетках.

Клетки ломаной, на которых их нет – серые. Белые клетки – не принадлежат ломаной и не содержат яблок. Всего на поле 89 яблок.

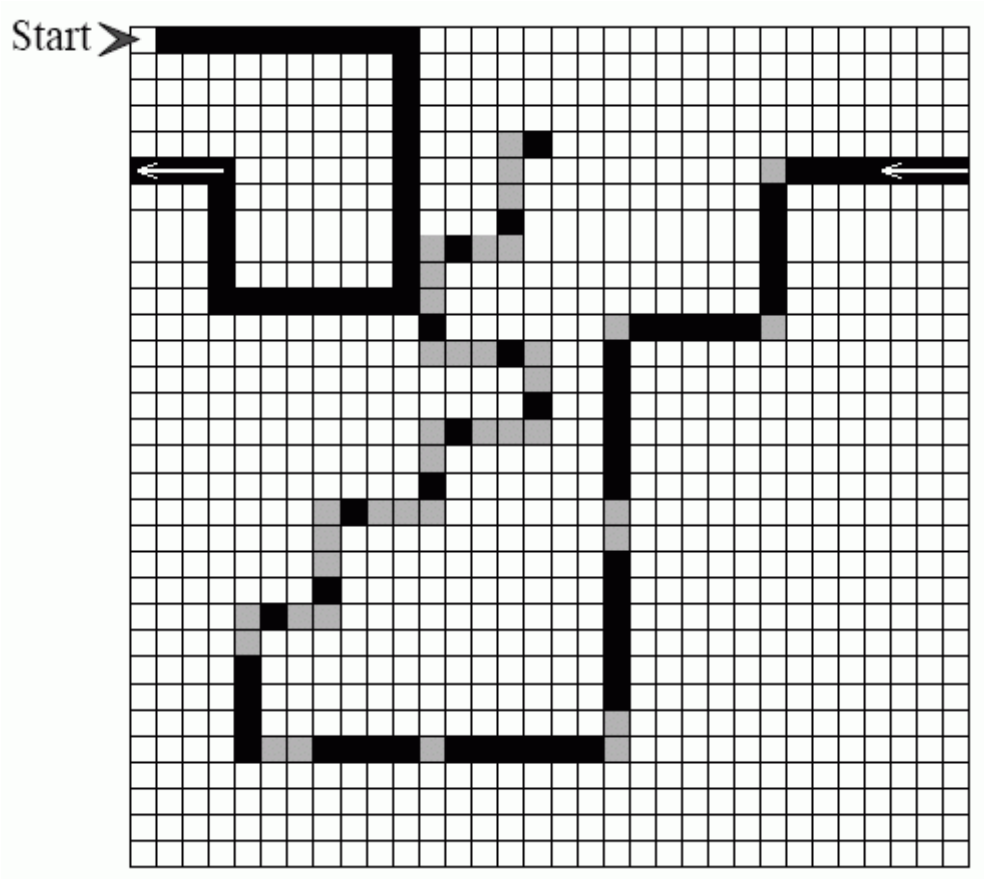


Рис. 60. Рис. Поле с яблоками

В клетке с пометкой “Start” находится муравей. Он занимает клетку

поля и смотрит в одном из четырех направлений (север, запад, юг, восток). В начале игры муравей смотрит на восток. Он умеет определять находится ли яблоко непосредственно перед ним. За один ход муравей совершает одно из четырех действий:

1. идет вперед на одну клетку, съедая яблоко, если оно было перед ним;
2. поворачивается вправо;
3. поворачивается влево;
4. стоит на месте.

Съеденные муравьем яблоки не восполняются. Муравей жив на всем протяжении игры – еда не является необходимым ресурсом для его существования. Никаких других персонажей, кроме муравья, на поле нет.

Ломаная строго задана. Муравей может ходить по любым клеткам поля.

На рис. изображены муравей, яблоки и часть поля.

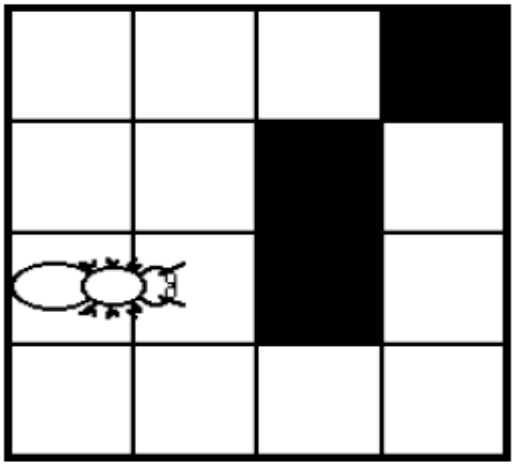


Рис. 61. Муравей и яблоки

Цель игры:

Игра длится 200 ходов, на каждом из которых муравей совершает одно из четырех описанных выше действий. В конце игры подсчитывается количество яблок, съеденных муравьем. Это значение – результат игры.

Цель игры – создать муравья, который за 200 ходов съест как можно больше яблок. Муравьи, съевшие одинаковое количество яблок, заканчивают игру с одинаковым результатом вне зависимости от числа ходов, затраченных каждым из них на процесс еды.

Однако эта задача может иметь различные модификации, например, такую, в которой при одинаковом количестве съеденных яблок, лучшим считается муравей, съевший яблоки за меньшее число ходов.

Ниже будет показано, что поведение муравья может быть задано конечным автоматом. При этом может быть поставлена задача о построении автомата с минимальным числом состояний для муравья, съедающего все яблоки, или автомата для муравья, съедающего максимальное количество яблок при заданном числе состояний.

Ввиду фиксированной топологии и размера поля (тор размером 32 на 32), фиксированного расположения яблок и их невосполнимости, а также примитивности поведения муравья, может показаться, что задача тривиальна.

Для того чтобы убедиться, что это не так, предлагается сконструировать муравья, который съест хотя бы 82 яблока за 200 ходов.

Приведенное выше число 82 выбрано неслучайно, так как существует простая стратегия с результатом 81.

Дж. Коза разработал общий подход для формального описания различных предметных областей, позволяющий посредством методов ГП вести поиск путей совершенствования реальных технических систем.

К настоящему времени "искусственные организмы", созданные Дж. Коза, успели найти несколько оригинальных решений, два из которых уже защищены патентами.

Ими стали

- новый метод определения оптимальной конфигурации проволочных антенн, обладающих требуемыми электромагнитными характеристиками;
- а также конструкция крыльев большого самолета, вызвавшая интерес компании Boeing.

Важным преимуществом ГП искусственного интеллекта Дж. Коза считает свободу от стереотипов и предубеждений, довлеющих над воображением разработчиков.

Нейронные сети.

Среди направлений работ в области ИИ следует также выделить НЕЙРОКИБЕРНЕТИКУ, или иначе говоря, подход к разработке машин, демонстрирующих «разумное» поведение, на основе архитектур, напоминающих устройство мозга и называемых нейронными сетями (НС).

В 1942 году, когда Н. Винер определил концепции кибернетики, В. Мак-Каллок и В. Питс опубликовали первый фундаментальный труд по НС, где говорилось о том, что любое хорошо заданное отношение вход-выход может быть представлено в виде формальной НС.

У рядовых муравьев и пчел примерно 80 нейронов на особь (у царицы - 200-300 нейронов), у тараканов - 300 нейронов и эти существа показывают отличные адаптационные свойства в процессе эволюции. У человека число нейронов более 10^{10}.

Одна из ключевых особенностей нейронных сетей состоит в том, что они способны обучаться на основе опыта, полученного в обучающей среде.

В 1957 году Ф. Розенблат изобрел устройство для распознавания на основе НС - персептрон, который успешно различал буквы алфавита, хотя и отличался высокой чувствительностью к их написанию.

Нетрадиционные подходы.

Мы живем на пороге третьего тысячелетия и часто не знаем, что природой в нас заложены совершенные, феноменальные возможности и способности.

Современной наукой доказано, что все болезни и неспособности человека находятся в прямой зависимости от состояния развития структур головного мозга и гармонии энергетических процессов в организме человека и окружающей природы.

Ученые констатируют использование ресурсов головного мозга человека только на 4-6%, остальная часть остается невостребованной. Гармония отношения человека и природы нарушена.

В настоящее время существует огромное количество разного рода духовных практик и психотехник, дошедших до нас из древних источников и времен. Примерами являются ДЭИР Д.С.Верищагина и Метод академика В.М.Бронникова «Информационное Развитие Человека». Возможно, что дальнейшее развитие этих практик и психотехник позволит связать энергоинформационные процессы с вычислительными машинами и приведёт к повышению эффективности работы программистов. Примером является ПСИХОБИОКОМПЬЮТЕР Бронникова.

Д.С.Верищагин так охарактеризовал историю системы ДЭИР: «История *системы ДЭИР* (дальнейшего энергоинформационного развития) началась в 1982 году, когда в мой кабинет заглянул экстрасенс Петр Келдоровский, ходивший в то время в чине полковника, и принялся вызнавать, нет ли у меня желания оставить проект, которым занималась моя группа - «психотронное оружие». Это и был проект моей группы.

Что такое «психотронное оружие», оно же официально прекращенный проект СС 0709 «Дружба»?

Суть его состоит вот в чем: целенаправленное воздействие экстрасенса (или группы людей) может вызвать в сознании человека (или их группы) изменения, которые окажут влияние на их поведение.

Например, если у восточного диктатора, занятого переговорами с соседом, появится непрерывная головная боль (или головокружение), то, вероятнее всего, переговоры не будут доведены до конца или не принесут желаемого результата.

Согласованное воздействие группы экстрасенсов-силовиков может вызвать, скажем, сердечный приступ у престарелого главы государства или стимулировать развитие рака у человека помоложе.

Лаборатории, занимающиеся психотронными программами, есть в каждом государстве и являются неотъемлемой частью разведывательно-диверсионных учреждений.

Я и тогда считал и сейчас убежден в том, что наша страна находится под непрерывным давлением других государств, инстинктивно боящихся сильных русских.

Страну, чтобы ее не разворовали и не сделали чьим-то сырьевым придатком, нужно защищать.

Нет, меня утомило другое: во-первых, работа с чрезвычайно неуравновешенными людьми — природными экстрасенсами (видели бы вы истерики, что ежедневно закатывала группа телепатов!), во-вторых, непрерывное давление со стороны аналогичных групп противника (головные боли, непонятные и неприятные события, необходимость все время проверять готовность антителепатов, поддерживающих защиту..).

Я принял предложение возглавить административную часть нового проекта.

Новый проект назывался «Пастырь» и выполнялся по заказу непосредственно ЦК КПСС. Суть его была изложена всего на нескольких десятках листков и сводилась к следующему.

В нашу задачу входила разработка системы приемов, при помощи которых один человек мог бы управлять многими — но так, чтобы те ничего не заподозрили; создать такую систему приемов, которые были бы доступны для любого, даже совершенно неразвитого человека со средней или слабой энергетикой.

Мы вынуждены были все проверять на самих себе. Для того чтобы процедуры управления стали действенными, человек должен усилить собственную энергетику. Но этого не добиться, если не отключить ее раз и навсегда от посторонних управляющих влияний — то есть стать неуправляемым со стороны.

Полностью в соответствии с требованиями заказчика мы создали систему управления посторонними людьми, которая мгновенно становилась неотъемлемой частью навыков обучающегося.

По инициативе Келдоровского мы ввели в проект элементы, связанные с продлением жизни и сохранением здоровья. Нами двигало просто стремление к совершенству — ведь вопросы поддержания здоровья при умении управлять собственной энергетикой настолько элементарны, что никто из посвященных не озабочен этими проблемами всерьез! Но мы ориентировались на людей неподготовленных, которые придут после нас.

Мы разработали и исследовали систему подъема личного магнетизма — харизмы, заставляющей обычных людей стремиться к подчинению.

В 1988 году система ДЭИР окончательно оформилась в том виде, в котором мы ее вам представляем.

Между тем политическая ситуация в стране менялась день ото дня.

Власть слабела на глазах, и мы, не чувствуя более опасности для себя, в 1989 году объявили о завершении своей работы.

Гриф секретности с проекта был снят.

Сергей Десменцов, воспользовавшись неразберихой и войнами на территории бывшего Советского Союза, смог уехать в Америку... А я остался, потому что не могу жить не в родной стране. Но с людьми я больше почти не встречаюсь.

Кроме того, передо мной стоит еще одна задача. Однажды, когда мы осознали всю глубину разработанной нами системы ДЭИР и поняли, что эти знания слишком важны, чтобы их потерять, мы обменялись доверенностями, позволяющими любому из нас действовать от лица остальных. Мы не могли позволить обретенным знаниям, способным дать надежду всему человечеству, умереть вместе с нами.

Я, Дмитрий Верищагин, передаю эту рукопись для издания моему доверенному лицу, соавтору и ученику Кириллу Титову. Я действую от своего лица и от лица моих коллег, первооткрывателей системы ДЭИР -- Петра Келдоровского, Алексея Грыщака и Сергея Десменцова.

Те, кто станут нашими учениками, получат надежду. Они освободятся и обретут здоровье. Они будут жить дольше обычных людей, и им будет сопутствовать удача. Они смогут то, что недоступно обычному человеку.

Счастья всем вам!

Дмитрий Верищагин от себя лично и от лица
Петра Келдоровского,
Алексея Грыщака,
Сергея Десменцова.»

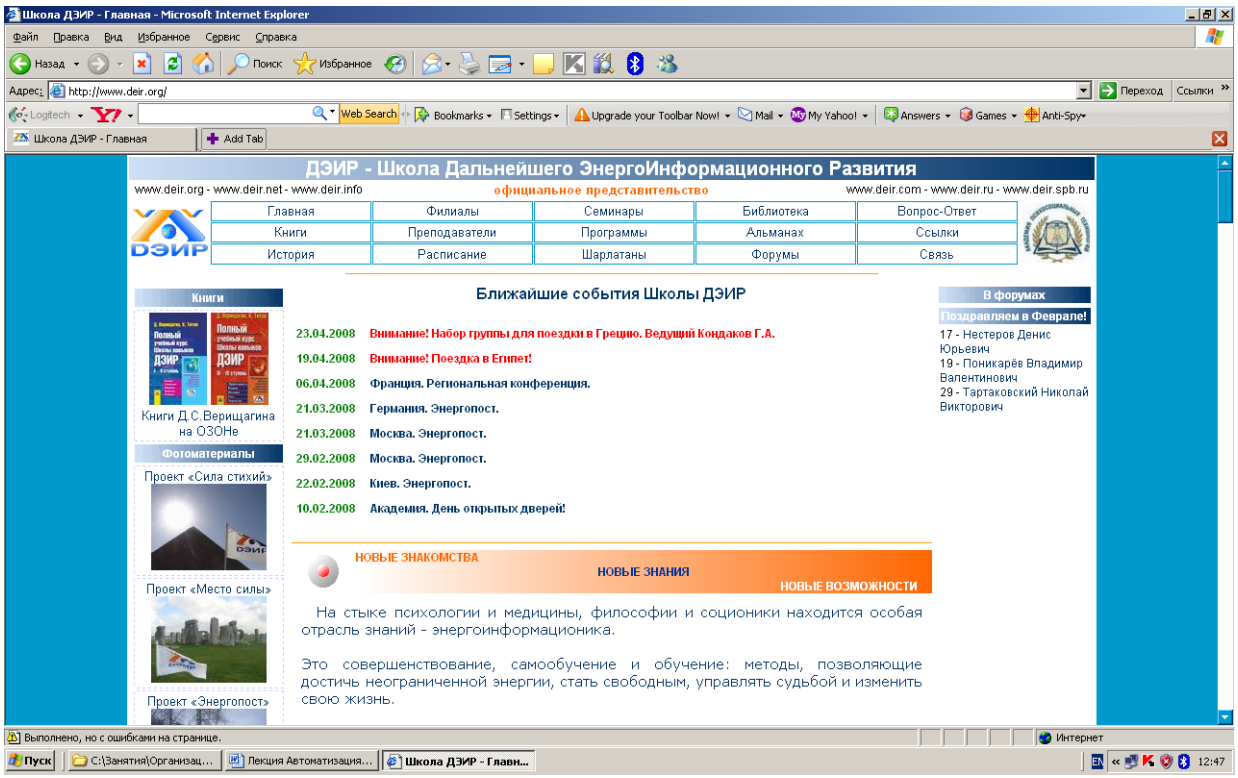


Рис. 62.

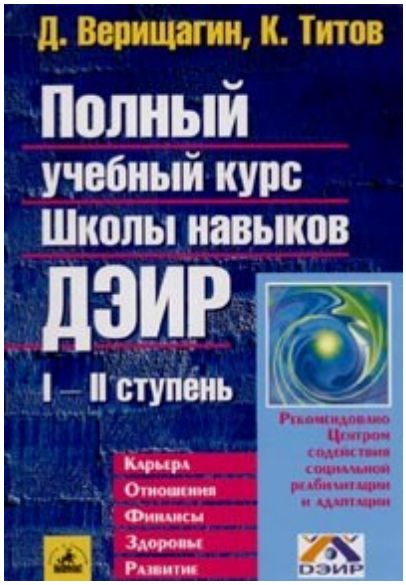


Рис. 63.

#

В.М.Бронников свои практики и психотехники изложил на сайте WWW.BRONNIKOV.RU - О Ф И Ц И А Л Ь Н Ы Й С А Й Т Московского Центра Развития Человека (Лицензия В.М.Бронникова серия МО № 00233 ООО "Информационные технологии")

Домашняя страница сайта "ИНФОРМАЦИОННОЕ РАЗВИТИЕ ЧЕЛОВЕКА" характеризует ПРОГРАММы ТРЕНИНГОВ:

I СТУПЕНЬ

РАЗВИТИЕ ФАНТОМНЫХ ЧУВСТВ

Рис. 64.

На наших семинарах и тренингах Вы сможете развить навыки естественного оздоровления, профилактики заболеваний и расширения диапазона чувствительности.



Рис. 65.

Развитие способности ощущать энергию своего организма и управлять своей жизненной силой, приобретение возможности сохранять активное состояние, что позволяет научиться не уставать.

Общее оздоровление, активизация и увеличение защитных функций организма – учим не болеть, обходиться без лекарств и помогать себе в сложных ситуациях используя упражнения I ступени.

Овладение технологией противодействия негативным энергоинформационным влияниям: людей, технических средств и среды.

Развитие нового уровня чувств и творческих способностей.

Организм человека приобретает новые защитные свойства и качества для существования в агрессивной энергоинформационной среде.

I СТУПЕНЬ
2 уровень усложнения

Рис. 66.

СУЩНОСТНАЯ ПРИРОДА ЧЕЛОВЕКА

Развитие базовых свойств сущностной природы человека.

Овладение свойствами и качествами базовых сущностей Земли, Неба и Срединного состояния.

Включение и развитие первородных животных качеств: змеи – осязание, тигра – обоняние, обезьяны – слуха, птицы – зрения.

В динамике, на основе техники «Прокачки», развитие качеств четырех зверей.

Развитие в человеке симбиоза чувств и качеств пяти видов «Дракона».

Наработка срединного качества между «Учеником» и «Учителем» по технике «Беременные животные».

Изученные упражнения курса - дадут Вам возможность раскрытия и включения дремлющих и невостребованных свойств и качеств сущностной природы человека. Учащийся получает совершенно новые, несвойственные обычному человеку ощущения своего внутреннего мира и окружающего пространства. Приобретает более широкий диапазон чувствительности, новые силы и свойства в организме. Человек начинает шире видеть и осознавать.

Занятия ведут к формированию в человеке духовной зрелости. Береженого Бог бережет. Появляются сущностные силы охраняющие, оберегающие человека, создающие по-новому его удачу, успех и счастье в жизни.

II СТУПЕНЬ

Рис. 67.

ВНУТРЕННЕЕ ВИДЕНИЕ

Развитие навыков сверхсознательной функции - ПСИХОБИОКОМПЬЮТЕР



Рис. 68.

Развитие способности сознательно, по команде создавать экран внутреннего видения (с закрытыми глазами), управлять мысленно экраном и изображением. Эта способность аналогична возможностям персонального компьютера.

Овладение способностью видеть свой организм изнутри.

Учит создавать информационную базу данных, приобретать новые виды памяти: психобиокомпьютерную, фотографическую (за доли секунды фиксировать на экране большой объем информации). Эти способности значительно улучшают процесс обучения. Развитый психобиокомпьютер выполняет функцию многих приборов (часов, компаса, бинокля, блокнота, персонального секретаря и т.д.).

Закладывается в психобиокомпьютер, и хранится информация по всем учебным предметам без какой-либо нагрузки. Это открывает новые оперативные возможности необходимые лидеру – Человеку Сознательному.

Видеть внутри себя энергоинформационные структуры, убирать негативные воздействия и формировать новые защитные свойства для противодействия.

Обучение видеть внутри себя в шесть направлений одновременно, то есть голографически, развивая многофункциональные возможности мозга.

Формирование личных свойств и качеств психобиокомпьютера.

Эти возможности развивают невостребованные свойства, качества и способности мозга человека.

III СТУПЕНЬ

Рис. 69.

ПРЯМОЕ ВИДЕНИЕ ЧЕЛОВЕКА

Развитие способностей внешнего видения с закрытыми глазами



Рис. 70.

Внешнее видение открывает перспективы для проектирования своего здоровья.

Прямое видение дает возможность ориентироваться в пространстве без помощи глаз, видеть в темноте, увеличивать микро объекты и приближать удаленные объекты. С закрытыми глазами читать книги, рисовать, кататься на роликах и т.д. Эти способности необходимы для наработки объективности работы психобиокомпьютера и возможности дальнейшего его развития.

Позволяет видеть внутренние материальные и энергоинформационные структуры людей и объектов.

Овладение технологией самопрограммирования мозга.

Использование системно-целостной психологии для работы с информацией в пространстве и во времени.

Путь развития от Человека Разумного, к Человеку Сознательному осуществляется на основе принципа: «от прямого видения – к прямым знаниям».

Базовые учебники

- 1. Бек К. Экстремальное программирование: разработка через тестирование. Библиотека программиста. – СПб.: Питер, 2003. – 224 с.: ил. ISBN 5-8046-0051-6. Главы 1 – 32.
- 2. Амблер С. Гибкие технологии: экстремальное программирование и унифицированный процесс разработки. Библиотека программиста. – СПб.: Питер, 2005. – 412 с.: ил. ISBN 5-94723-545-5. Главы 1 - 30.
- 3. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник. - 2-е изд., перераб. и доп. - М.:Финансы и статистика, 2006.-544с.: ил. Главы 1, 4, 5, 7.
- 4. Кириченко А.А. Комплексирование программных средств: использование в C#-программах системы команд и программ ОС Windows. М. ООО ИПЦ «МАСКА», 2010.
- 5. Кулямин В.В. Технологии программирования. Компонентный подход. М., Бином, 2007, www.intuit.ru.
- 6. Кариев Ч.А. Разработка Windows-приложений на основе Visual C#. М., Бином, 2007, www.intuit.ru.
- 7. Кен Ауер, Рой Миллер. Экстремальное программирование: постановка процесса. С первых шагов и до победного конца..- Спб.: Питер, 2004.-368с., ISBN 5-318-00132-7. Главы 10-14.
- 8. Астелз , Дэвид ; Миллер. Гренвил; Новак. Мирослав. Практическое руководство по экстремальному программированию.: Пер. с англ.-М.: Издательский дом “Вильямс”, 2002.-320с.. ISBN 5-8459-0329-7. Главы 1-18.

- 9. Кириченко А. А. Администрирование компьютерных сетей. Сетевое электронное учебное пособие. М. : НИУ ВШЭ, 2014. http://www.hse.ru/pubs/share/direct/document/125422665
- 10. Кириченко А.А. Архитектурные особенности различных стилей программирования. М.: ООО «ИПЦ Маска», 2012.
- 11. Фролов А.В., Фролов Г.В. Визуальное проектирование приложений C#. М., Кудиц-образ, 2003.

К настоящему времени помимо книг существует большое количество web-ресурсов по общим вопросам XP http://www.xprogramming.com/

- xProgramming.com
- extremeProgramming.org.
- http://www.xprogramming.ru.

Вопросы тестирования обсуждаются на сайтах

- Библиотека для тестирования Java-программ JUnit: http://www.junit.org
- Набор библиотек для тестирования под многие языки программирования: http://www.xprogramming.com/software.htm
- Списки рассылки по вопросам тестирования и рефакторинга: http://groups.yahoo.com/group/junit
  - http://groups.yahoo.com/group/junit http://groups.yahoo.com/group/testdrivendevelopment
  - http://groups.yahoo.com/group/testdrivendevelopment http://groups.yahoo.com/group/refactoring
  - http://groups.yahoo.com/group/refactoring

Приложение 1. TeamSpeak.

Описание функций и настройки клиентской части программы TeamSpeak.

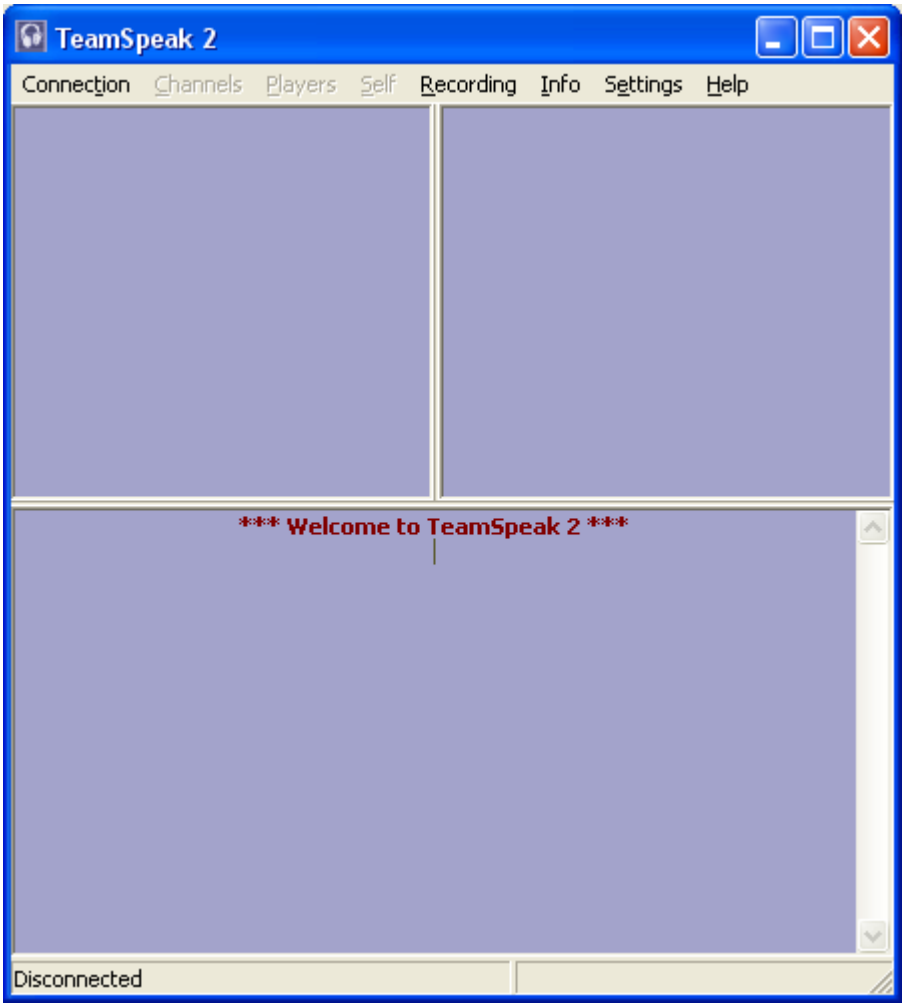


Рис. 71.

Откройте в меню **Settings (Настройки) -> Sound Input/Output Settings (Входящие \ исходящие настройки звука)**, в появившемся окне, в поле **Voice Send Method (Способ звукового послания)**, установите переключатель в положение **Push to talk (Нажать, чтобы говорить)**. Затем нажмите кнопку **Set (Изменить)** и после этого, клавишу на клавиатуре (не рекомендуется ставить клавишу **Alt, Ctrl, Shift**, т.к. используется во многих сочетания раскладки языков и переключения окон), с помощью, которой вы бы хотели включать исходящую передачу звука с микрофона на сервер **TeamSpeak**.

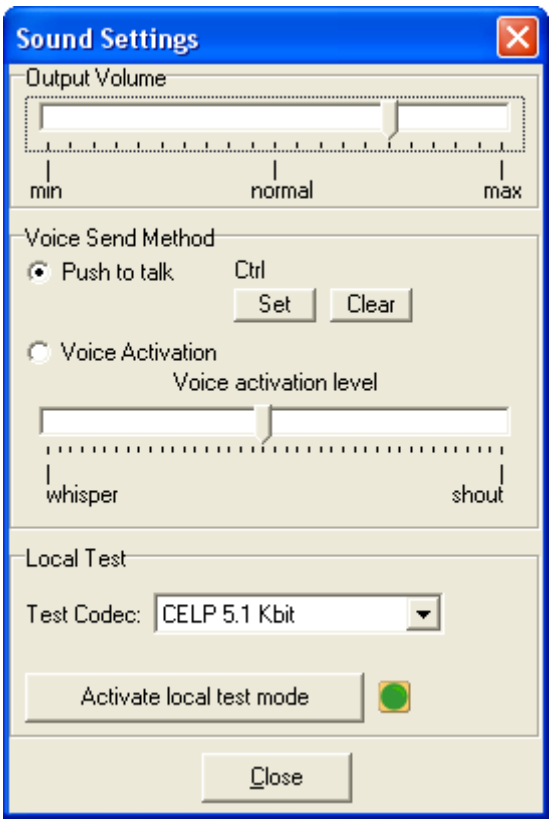


Рис. 72.

Если вы хотите, чтобы передача включалась автоматически при увеличении уровня звука, оставьте переключатель в положении **Voice Activation (Активация звука).** (Не рекомендуется, т.к. при неправильной настройке "обрезает" начало фразы или включается, когда это не требуется, от какого-либо звука). Настройте параметр **Voice Activation Level (Уровень активации звука) Whisper** - тише, **Shout** - громче так, чтобы передача включалась только тогда, когда вы говорите, а не была постоянно включена или не включалось от щелканья клавиатуры, вентилятора компьютера или помех, вызванных вибрацией подставки микрофона... Проверить настройку передачи звука можно с помощью кнопки **Activate Local Test Mode (Активировать тестовый режим)**, предварительно выставив в ниспадающем меню аудио кодек, при помощи которого будет производиться проверка. В процессе проверки вы будете слышать себя так, как будут слышать вас остальные пользователи на сервере (не забудьте только выключить воспроизведение с микрофона в микшере, иначе будете слышать себя 2 раза). Параметр **Output Volume (Исходящий звук)** регулирует громкость самой программы **TeamSpeak**, а не громкость, с которой передается ваш голос на сервер. Будьте внимательны, этот ползунок позволяет решить проблемы с перегрузкой звука, приходящего с сервера.

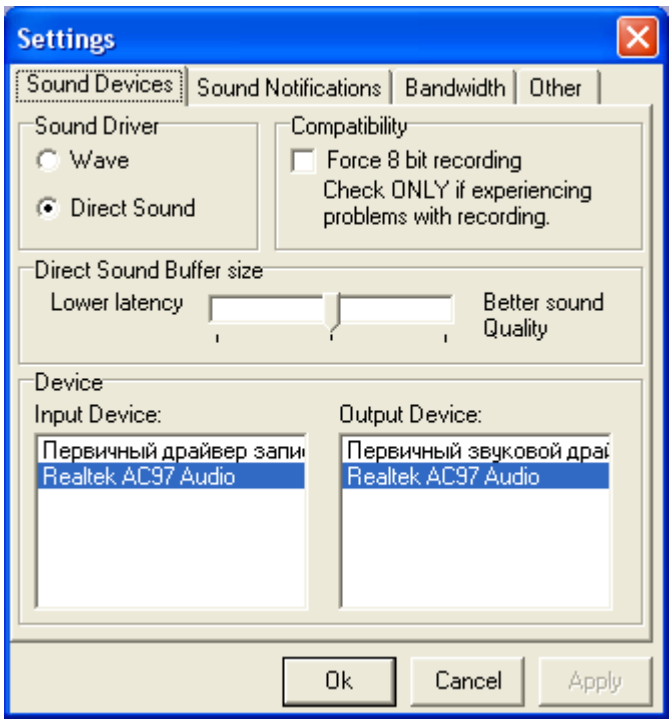


Рис. 73.

В меню **Settings (Настройки)** -> **Options (Опции),** вы увидите 4 вкладки: на первой вкладке - **Sound Devices (Звуковые устройства)** в поле **Sound Driver (Звуковой драйвер)** выберите **Wave (Стандартный)**, чтобы воспользоваться драйвером вашей звуковой карты (разумеется, если у вас хорошая звуковая карта) или выберите **Direct Sound (Улучшенный)**, чтобы воспользоваться драйвером **DirectX(9x)**. Далее - поле **Direct Sound Buffer Size (Размер буфера под звук)**. В этом поле можно установить размер буфера **Direct Sound (Улучшенный)**: чем он больше, тем качественнее звук, чем меньше, - тем меньше задержка на обработку звука. Ниже находится поле **Device (Оборудование)**, в нем вы можете выбрать устройства для ввода (слева) и для вывода (справа) звука на вашем компьютере. Причем, **"Первичный звуковой драйвер"** и **"Первичный драйвер записи"** это устройства, установленные как **"Основные устройства"** в разделе Панели управления **Windows "Звук и Мультимедиа"** на вкладке **"Аудио"**. Вкладка **Sound Notifications (Звуковые сообщения),** посвящена информационным звуковым сообщениям программы и их настройке, поле **Action (Действие)** позволяет выбрать настраиваемое событие.

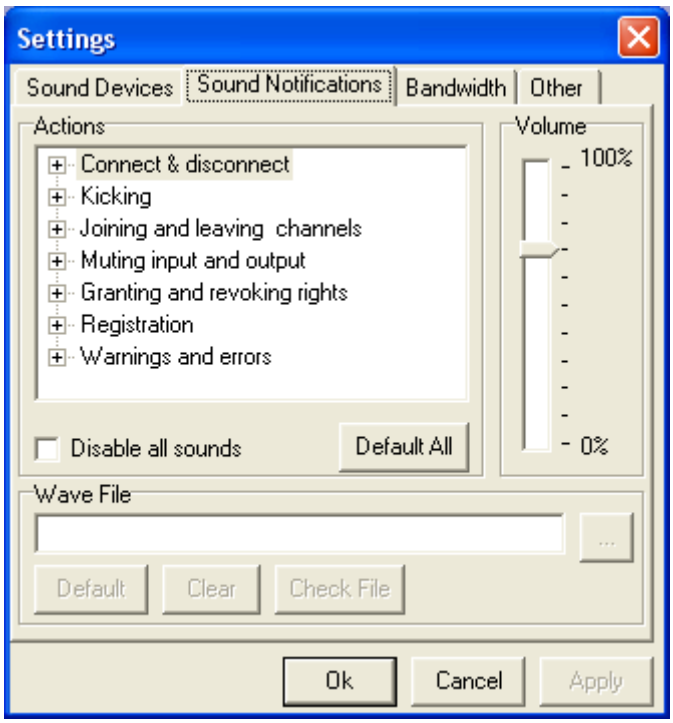


Рис. 74.

На второй вкладке **Sound Notifications (Звуковые сообщения)** вы можете отредактировать звуковые уведомления, возникающие в тех или иных случаях рассмотренных ниже:

Список событий:

Connect&disconnect:

- **Connected** - Звуковое сообщение после установления соединения с сервером
- **Disconnected** - то же самое, но после отсоединения
- **Connection lost** - при потере связи с сервером

Kicking:

- **Kick and diskonnected** - когда вас выкинули с сервера
- **Player kicked** - когда кого-то выкинули с сервера при вас

Присоединение (отсоединение) от каналов:

- **Swiched channel** - после перехода из одного канала в другой

- **Player joined** - когда к каналу, в котором вы находитесь, подсоединился незарегистрированный пользователь
- **Player left** - когда незарегистрированный пользователь покинул канал
- **Registered Player Joined** - когда к каналу присоединился зарегистрированный пользователь
- **Registered Player left** - когда зарегистрированный пользователь покинул канал

Включение и отключение микрофона:

- **Microphone muted** - после программного отключения микрофона в **TeamSpeak**
- **Microphone activated** - после программного включения микрофона
- **Sound muted** - после программного выключения звука в **TeamSpeak**
- **Sound activated** - после программного включения звука

Получение и запросы на права:

- **Voice granted** - при получении права голоса на сервере (в канале)
- **Voice revoked** - при снятии права голоса на сервере (в канале)
- **Voice requested** - при запросе на получение права голоса
- **Operator granted** - при получении прав оператора
- **Operator revoked** - при снятии прав оператора
- **Admin granted** - при получении прав администратора канала
- **Admin revoked** - при снятии прав администратора канала
- **Server admin granted** - при получении прав администратора сервера
- **Server admin revoked** - при снятии прав администратора сервера

Регистрация:

- **Registration complete** - при удачном завершении регистрации
- **Registration failed** - при неудачном завершении регистрации

Предупреждения и ошибки:

- **Incorrect password** - после ввода неверного пароля
- **Acces denied** - доступ запрещён
- **Warning** - при предупреждении о чем-либо
- **Error –** при ошибке

При помощи поля **Volume (Громкость)** можно регулировать громкость всех сообщений программы, параметр **Disable all sounds (Запретить все звуки)** позволяет отключать все голосовые сообщения вообще, а кнопка **Default All (Исходные параметры)** возвращает значения "по умолчанию" для всех событий. Если вы хотите сами выбрать звуковые файлы, при помощи которых программа будет сообщать вам о происшествии событий, изложенных в списке, то к вашим услугам поле **Wave file (Звуковые файлы)**. Порядок действий следующий: в поле **Actions (Действие)** выбрать событие, затем в окно для заполнения поля **Wave file (Звуковые файлы),** ввести адрес файла, который вы хотите использовать для озвучивания данного события (с помощью кнопки справа, с тремя точками, можно просмотреть содержимое дисков и найти файл средствами browser'a). Файл можно проверить, нажав кнопку **Check file (Проверить файл)**, - если всё в порядке, появится диалоговое окно **"File is ok"(Всё работает исправно)** (желательно использовать при создании своих файлов (или приспосабливании имеющихся) те же звуковые параметры (имеющиеся файлы можно конвертировать программой для обработки звука), что у файлов, предлагаемых программой по умолчанию). Кнопка **Clear (Очистить)** очищает поле ввода, а кнопка **Default (Исходные),** устанавливает значение "по умолчанию" для данного события.

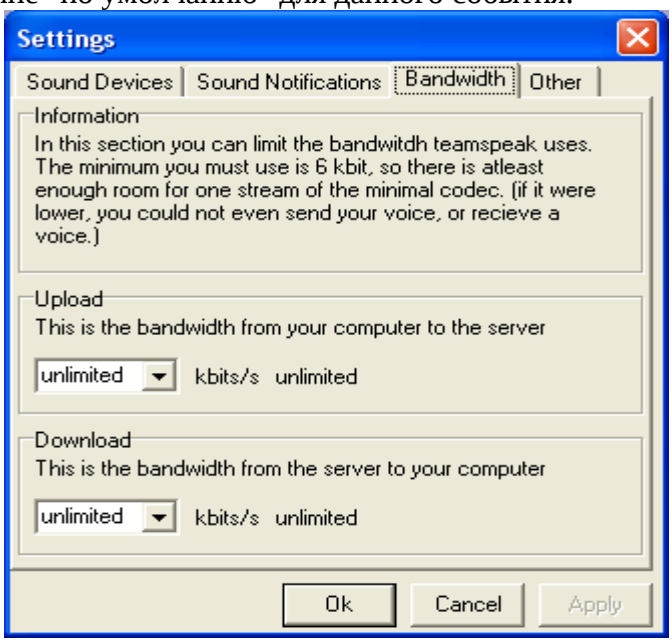


Рис. 75.

Вкладка **Bandwidth (Пропускная способность)** позволяет ограничить использование канала связи программой. В поле **Information (Информация)** содержится примерно следующий текст: "Здесь вы можете ограничить ширину канала для программы **Teamspeak**. Минимальное значение, которое вы можете установить - **6 Кбит**, это значение позволяет транслировать 1 поток аудиоданных в минимальном кодеке. (Если бы значение было меньшим, то передача звука была бы невозможна.)" Далее вы видите 2 поля: **Upload (Отправленные)** и **Download (Принятые)**, выпадающие списки в них устанавливают

параметры ограничения для отправки на сервер и принятия с сервера аудиоданных соответственно.

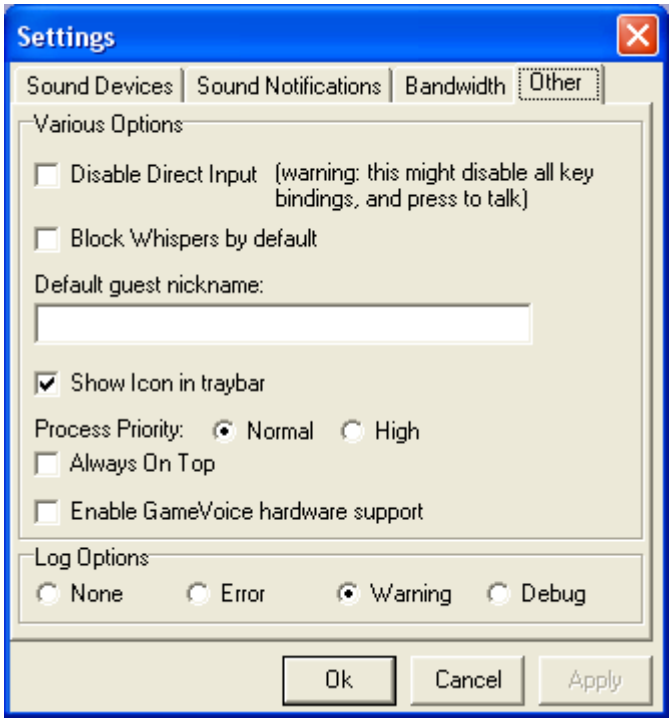


Рис. 76.

Последняя вкладка – **Other (Прочие)**. Обратим свое внимание на поле **Various Options: Disable Direct Input (Дополнительные опции: Запретить входящие)** - пометьте галочкой, если у вас не все в порядке с сочетаниями клавиш или использованием клавиатуры компьютера во время игры. Внимание, при включении этой опции также может не работать клавиша, установленная ранее для включения передачи звука на сервер и все сочетания клавиш, задействованные программой **Teamspeak**. **Block Whispers by Default (Блокировать Личные по умолчанию)** - при включении, будут блокироваться любые попытки приватного разговора с вами каждый раз, когда вы будете запускать **Teamspeak** и заходить на какой-либо сервер. **Default Guest nickname (Стандартные гостевые имена)** - имя пользователя, под которым вы будете входить на сервер, если вы не зарегистрированы на нем. **Show Icon in traybar (Показывать яярлик в нижней панели задач)** - если пометить галочкой, то, свернув окно программы, она окажется не на панели задач, а в нижней панеле задач, рядом с часами (очень удобно). **Process priority (Приоритеты)** - устанавливает приоритет программы для ОС можно поставить **High (Высокий)** или **Normal (Обычный)**, соответственно нужды программы в первом случае будут выполняться операционной системой вне очереди, а во втором приоритет выполнения будет таким же, как и у остальных приложений. Это необходимо, когда в процессе игры звук принимается с ошибками или кусками в связи с большой загрузкой жесткого диска компьютера - положение **High (Высокий)** позволяет устранить подобные проблемы. **Always On Top (Поверх остальных)** - окно программы всегда поверх остальных окон, даже когда оно не активно. Если вы являетесь обладателем **Game Voice Pack ()**, поставьте галочку перед **Enable Game Voice Hardware Support (Разрешать голос аппаратной поддержи)**. Программа **Teamspeak** может вести запись событий в **Log (Файлы (журналы))**, для настройки параметров ведения журнала специально сделано поле **Log options (Файлы настроек)**. **None** - не вести журнал вообще... **Error (Ошибка)** - записывать в журнал сообщения об ошибках, **Warning (Предупреждение)** - вести запись ошибок и предупреждений, **Debug (Исправлять)** - записывать все события. Если возникнут неполадки с программой, то журнал **Log** (**Файл**) можно отправить по

электронной почте службе технической поддержки **Teamspeak**, и, при помощи данных журнала, специалисты, возможно, смогут помочь устранить их.

После того, как все настройки завершены, запустим **Connection (Соединения)** -> **Connect (Присоединиться)**. Окно **Connect to Server (Присоединиться к серверу)** открывается на вкладке **Local adressbook (Локальная адресная книга)**. Это ваша записная книжка - сюда можно заносить адреса понравившихся вам серверов **TeamSpeak**. Сейчас она пуста, поэтому переходим на вкладку **Web server list (Сервер лист)**. Программа сразу предложит вам установить фильтр для поиска интересующих вас серверов (если программа не предлагает установить фильтр и в последующем окно установки фильтра можно вызвать кнопкой **Change filter (Изменить фильтр)**).

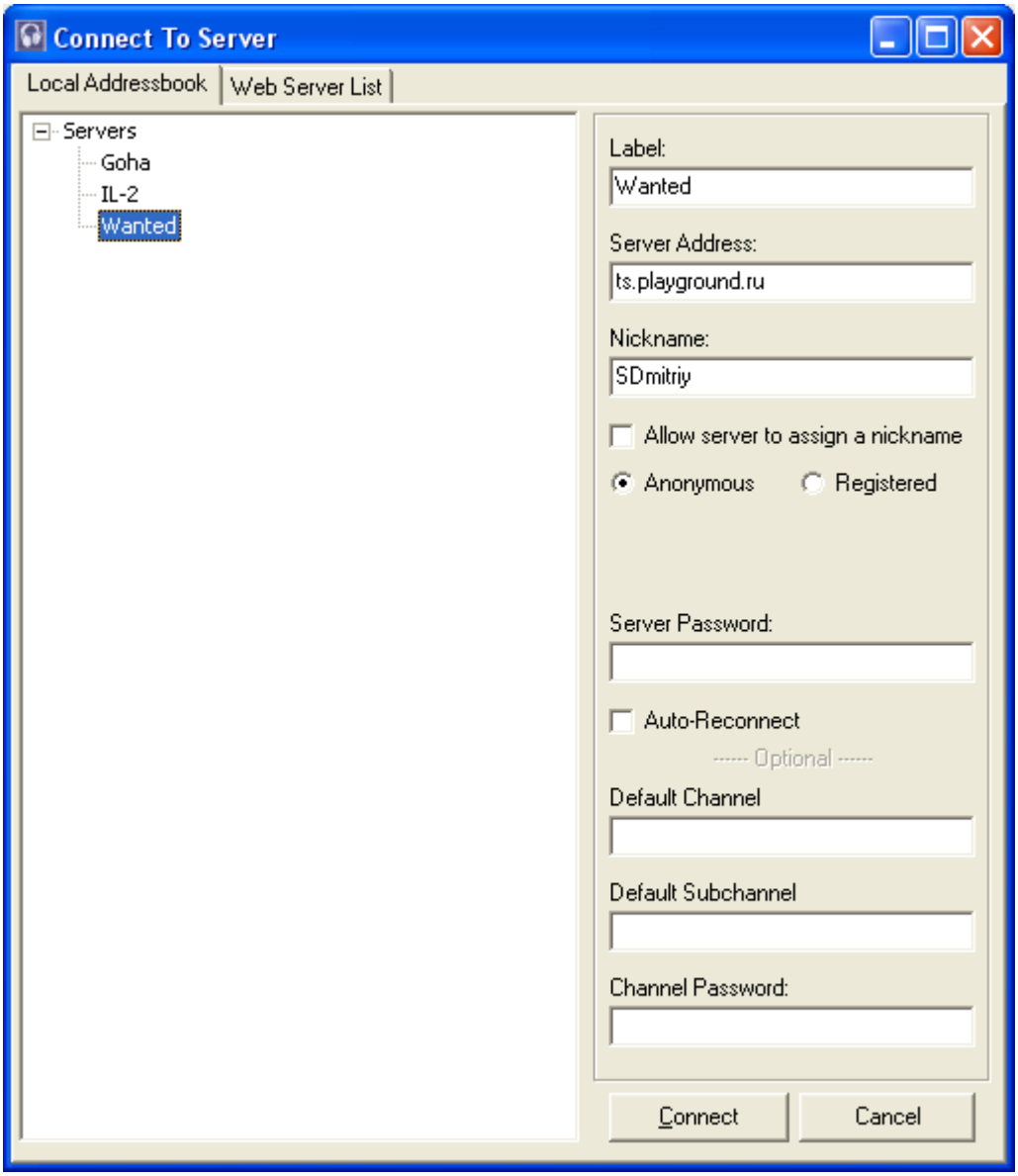


Рис. 77.

Окно **Server List Filter (Фильтр серверного листа)** позволяет вам выбрать параметры, по которым Teamspeak будет искать серверы в интернете. В поле **Server type filter (Тип серверного фильтра)** можно выбрать, какой тип серверов будет искать программа:

- **Show all** - фильтр выключен,
- **Show clan servers only** - только серверы кланов,
- **Show public server only** - только публичные серверы.

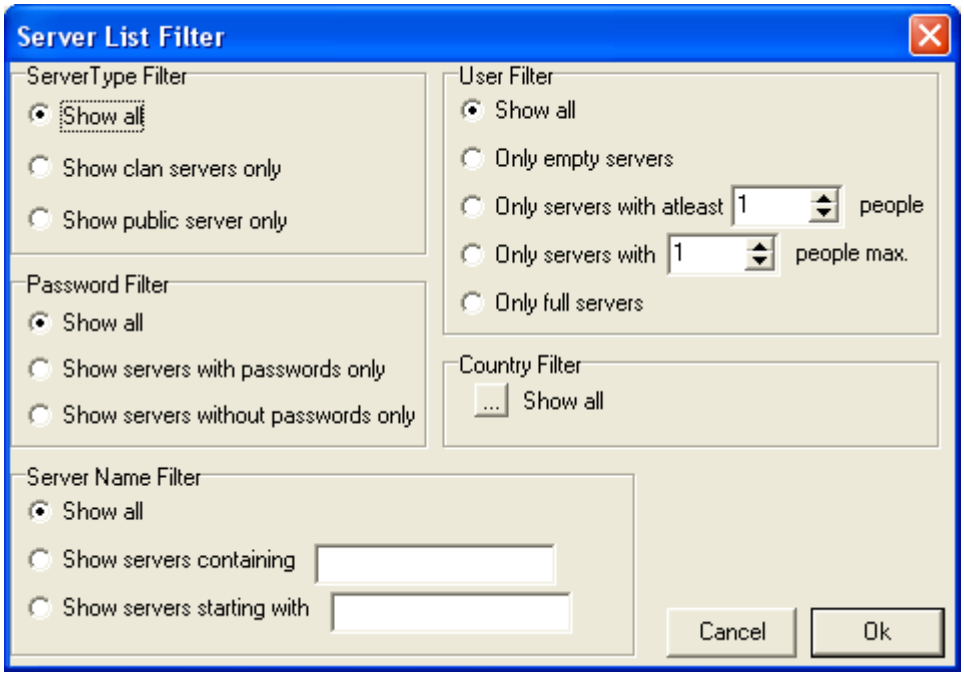


Рис. 78.

Поле **Password filter ()** позволяет:

1. искать серверы независимо от наличия или отсутствия пароля (**Show all**),
2. искать только запароленные серверы (**Show servers with passwords only**) или только серверы без пароля (**Show servers without passwords only**).

Server name filter позволяет установить фильтр на имена серверов:

- **Show all** - фильтр выключен,
- **Show servers containing** - программа будет искать серверы с именами, содержащими в названии то, что вы ввели в окне для ввода справа,
- **Show servers starting with** - **Teamspeak** будет искать серверы с именами, начинающимися с того, что вы введете в окне справа...

В поле **User filter** выбираете, какие серверы вам нужны относительно количества пользователей:

- **Only empty servers** - искать только пустые,
- **Only servers with at least * people** - будет произведен поиск всех серверов, на которых есть как минимум * пользователей (кол-во пользователей вводите в соответствующее окно),
- **Only servers with * max** - будет произведен поиск серверов, к которым присоединено определенное количество указанных вами в окне рядом количества пользователей,

- **Only full servers** - сервера, все места на которых заняты (и зачем такие искать?).
- **Country filter** – фильтр, позволяющий оградить себя от поиска иностранных серверов (конечно, если вы не полиглот).

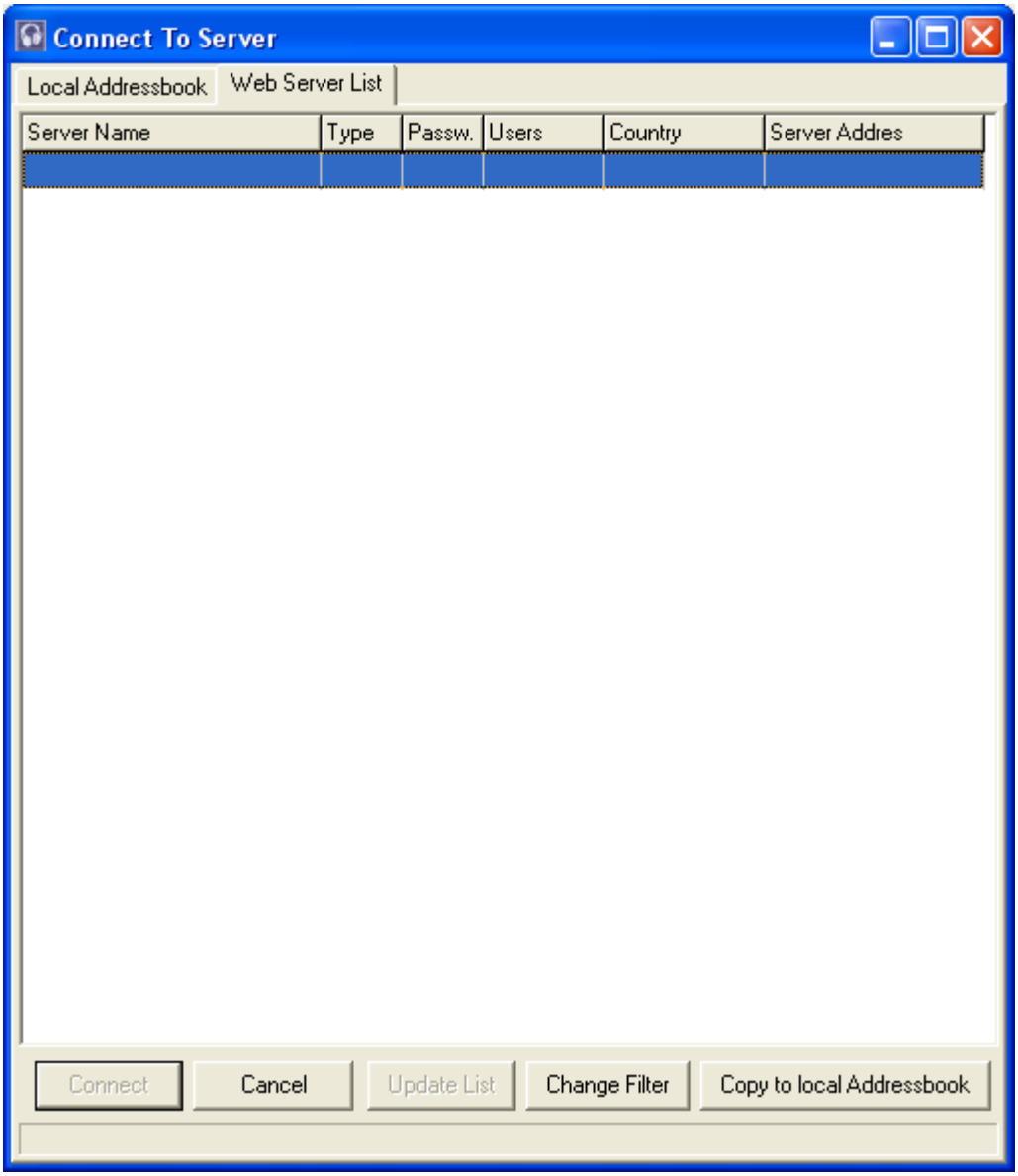


Рис. 79.

Имея список найденных по фильтру серверов, можно присоединиться к любому из них, выбрав его в списке и нажав кнопку **Connect (Подключения)** в левом нижнем углу окна. Чтобы обновить информацию по найденным серверам, нужно нажать кнопку **Update List (Обновить лист)**, чтобы добавить сервер в свою адресную книгу (что на соседней вкладке) - **Copy to Local adressbook (Скопировать в локальную адресную книгу).** Если ничего не хотите делать со списком, то можно либо задать другой фильтр (кнопка **Change Filter (Изменить фильтры)**), либо просто закрыть окно (кнопка **Cancel (Отменить)**).

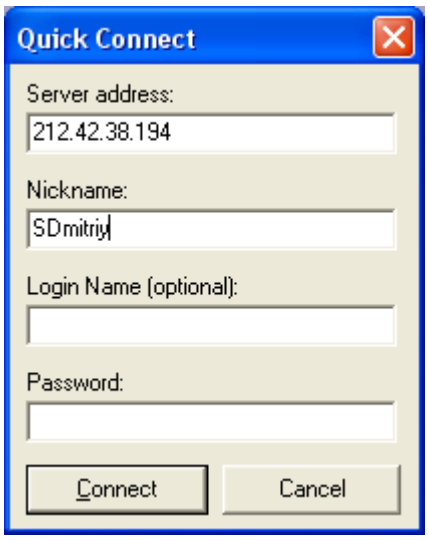


Рис. 80.

Запустите **Connection (Подключения)** -> **Quick connect (Быстро подключться)**. Появится окно - **Quick Connect connect (Быстро подключться)**, с его помощью, можно быстро присоединиться к серверу по его IP или имени вида **voice.teamspeak.org:9500**. В верхнем поле вводим соответственно IP (**Например: 212.42.38.194**), затем, в поле **Nickname** - свой ник, под которым будет производиться заход на сервер. Если вы указали в поле адрес, тот адрес, что был дан для примера, то вы попадете на сервер **www.Playground.ru**. Слева вы увидите список всех каналов (комнат) на сервере, справа - краткую информацию о сервере, канале и о пользователе, выбранном вами в левом окне.

Нижнее окно - консоль сервера. Даже если вы один на сервере, а уж тем более, если не один, то наверняка заметите, что справа от **Nickname** каждого пользователя, или справа от названия каждого канала, в скобках, будут расположены буквы и (или) сочетания букв - это флаги. Флаги оповещают всех пользователей о том, какими правами обладает пользователь, к которому относятся флаги, или какие установки стоят на комнате, которой эти флаги принадлежат.

Для пользователей:

- **U - unregistered (Anonymous)** незарегистрированный пользователь (обычно вы в первый раз заходите на новый сервер именно с таким флагом)
- **R – Registered** зарегистрированный пользователь
- **V - Voiced** пользователь обладает правом голоса - т.е. пользователь может разговаривать с другими в комнатах с флагом **M**,
- **AV - Auto-voiced** то же самое, но, если у вас нет флагов **V** или **AV**, то разговаривать в каналах, помеченных флагом **M** вы не сможете,
- **O(AO) - Channel Operator** Оператор канала,
- **CA - Channel admin** администратор канала,
- **SA - Server admin** администратор сервера.

Для комнат (каналов):

- **R - Registered** После создания, канал остается на сервере, даже если его покидают все пользователи (при отсутствии этого флага, когда все пользователи покидают канал - канал удаляется с сервера),
- **U - Unregistered** Незарегистрированный канал.
- **M - Moderated** В таких каналах разговаривать могут только пользователи с флагами **V** и **AV**, остальные могут просто прослушивать канал...
- **S - Subchannels** в канале, помеченном данным флагом, пользователи, могут создавать подканалы.
- **D - Default** в канал, помеченный данным флагом приходят по умолчанию пользователи после установки связи с сервером.
- **P - Passworded** Канал защищен паролем.

На каждом сервере, права для каждого флага и типа пользователей свои... о правах каждого флага на конкретном сервере можно узнать, запустив в меню тимспика **Info (Информация)-> Show Premissions (Показывать права)**, здесь, сверху можно выбрать тип пользователя, а снизу будут показаны его права... для удобства восприятия права делятся на 4 группы: **Administration (Администраторская), Channel Premissions (Права на канале), Player Priveleges (Персональные привелегии)** и **Other (Прочие)**. Соответственно, если отметка - красный крест, значит, таких прав этот тип пользователей не имеет, а если зеленая галочка - имеет.

Список и расшифровка пользовательских прав:

- **Send Text to everyone** - возможность посылать текстовые сообщения кому угодно.
- **Send Text to own channel** - возможность посылать текстовые сообщения в собственный канал (комнату).
- **Send Text to all channels** - возможность посылать текстовые сообщения во все каналы на сервере.
- **Send text to player** - возможность посылать текст пользователю.
- **Kick player from server** - право выкинуть (принудительно отсоединить) любого пользователя с сервера.
- **Kick player from channel** - право выкинуть любого пользователя из канала общения.
- **Allow Channel Commander** - право назначать Командира канала. (Модераторы каналов могут переговариваться друг с другом, находясь в разных каналах!)

Права пользователей:

- **Grant/Revoke - SA** право назначать / снимать администратора сервера.
- **Grant/Revoke - CA** право назначать / снимать администратора канала.
- **Grant/Revoke - OP** право назначать / снимать операторов и автооператоров канала.
- **Grant/Revoke Voice** - право давать / отнимать голос пользователям.
- **Grant/Revoke allow registration** - право разрешать / отменять разрешение на регистрацию отдельным пользователям.
- **Register player** - полномочия зарегистрировать пользователя.
- **Delete player registration** - полномочия удалить регистрацию пользователя.
- **Allow self-registration** - разрешить пользователю самостоятельно зарегистрироваться.

Разрешения на каналах:

- **Join Registered** - можно подсоединяться к каналу с флагом **R**
- **Create registered** - можно создавать каналы с флагом **R**
- **Create unregistered** - можно создавать каналы без флага **R**
- **Create default** - можно создавать каналы с флагом **D**
- **Create Sub-channeled** - можно создавать каналы с флагом **S**
- **Create Moderated** - можно создавать каналы с флагом **M**
- **Delete channel** - можно удалять каналы
- **Edit Name** - право на смену названия канала
- **Edit Password** - право на смену пароля канала
- **Edit topic** - право на смену темы канала
- **Edit description** - право на смену комментария к каналу
- **Edit order** - право менять очередность каналов на сервере.
- **Edit maxusers** - право изменять максимальное число пользователей для канала
- **Edit codec** - право менять кодек, используемый каналом.
- **Join without giving password** - право заходить на каналы, защищенные паролем (флаг канала **P**) без ввода пароля

Администрация:

- **Webadmin register player** - право зарегистрировать пользователя с правами администратора
- **List Registered Users** - право просматривать список зарегистрированных пользователей
- **Change own password** - право менять собственный пароль
- **Change all users passwords** - право менять пароли всех зарегистрированных пользователей
- **Set Premissions** - право раздавать права
- **List all servers** - право просматривать список всех серверов **TeamSpeak** на машине
- **Add server** - право добавить сервер **TeamSpeak**
- **Delete server** - право удалить сервер
- **Edit server port** - право менять порт сервера
- **Edit webpost url** - право менять url
- **Edit Server name** - право менять имя сервера
- **Edit server max users** - право менять максимально допустимое кол-во пользователей на сервере
- **Edit welcome message** - право менять содержание приветственного сообщения (появляется в консоли при заходе на сервер)
- **Edit server password** - право менять пароль сервера
- **Edit server type** - право менять тип сервера
- **Edit allowed codecs** - право выбирать, какие кодеки разрешаются к использованию на сервере
- **Ban ip** - право запретить пользователя по его IP
- **Move players between channels** - право перемещать пользователей из канала в канал
- **Stop the server** - право останавливать сервер
- **Start the server** - право запускать сервер

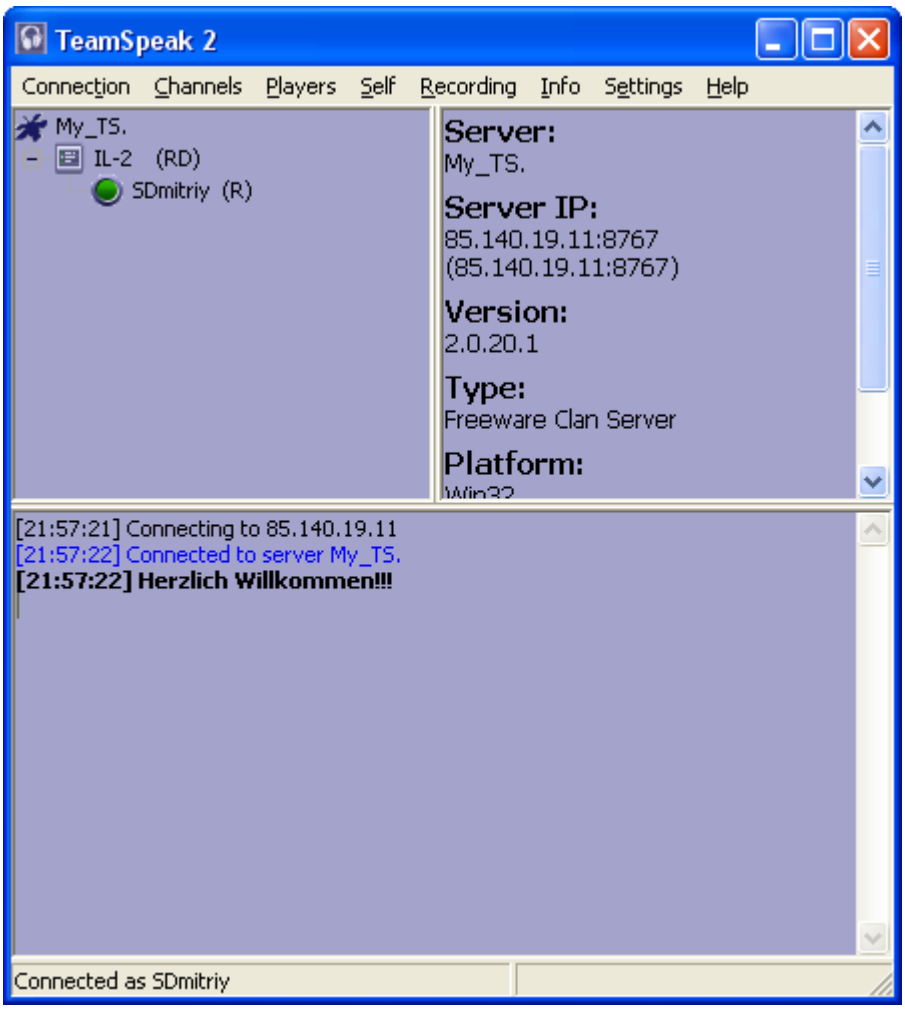


Рис. 81.

Регистрация на сервере. Находясь на сервере выбираем в меню программы: **Self (Собственные) -> Rregister With Server (Зарегистрироваться на сервере)**, открывается окно **Register With Server(Зарегистрироваться на сервере)** - в нем три поля в первом (**Login Name**) вводите свое имя, под которым вы хотите быть зарегистрированы на сервере. В поле **Password (Пароль)** введите ваш пароль, под которым вы впоследствии будете заходить на сервер как зарегистрированный пользователь, в поле **Password check (Проверять пароль)** повторите пароль, затем смело жмите **Register (Регистрировать)** и справа от вашего **Nickname** появится флаг **R**. Не забывайте заглядывать в **Info (Информация)-> Show Premissions (Показывать права)**, на некоторых серверах самостоятельная регистрация невозможна... (смотрите права для типа Anonimous). Если вы находитесь в канале с меткой **M**, то вам необходимо получить право голоса, для этого, опять же загляните в **Info (Информация)-> Show Premissions (показывать права)** и посмотрите, какой тип пользователей может вам его предоставить. Обычно право голоса могут давать администраторы сервера и канала (флаги **SA** и **CA** соответственно).

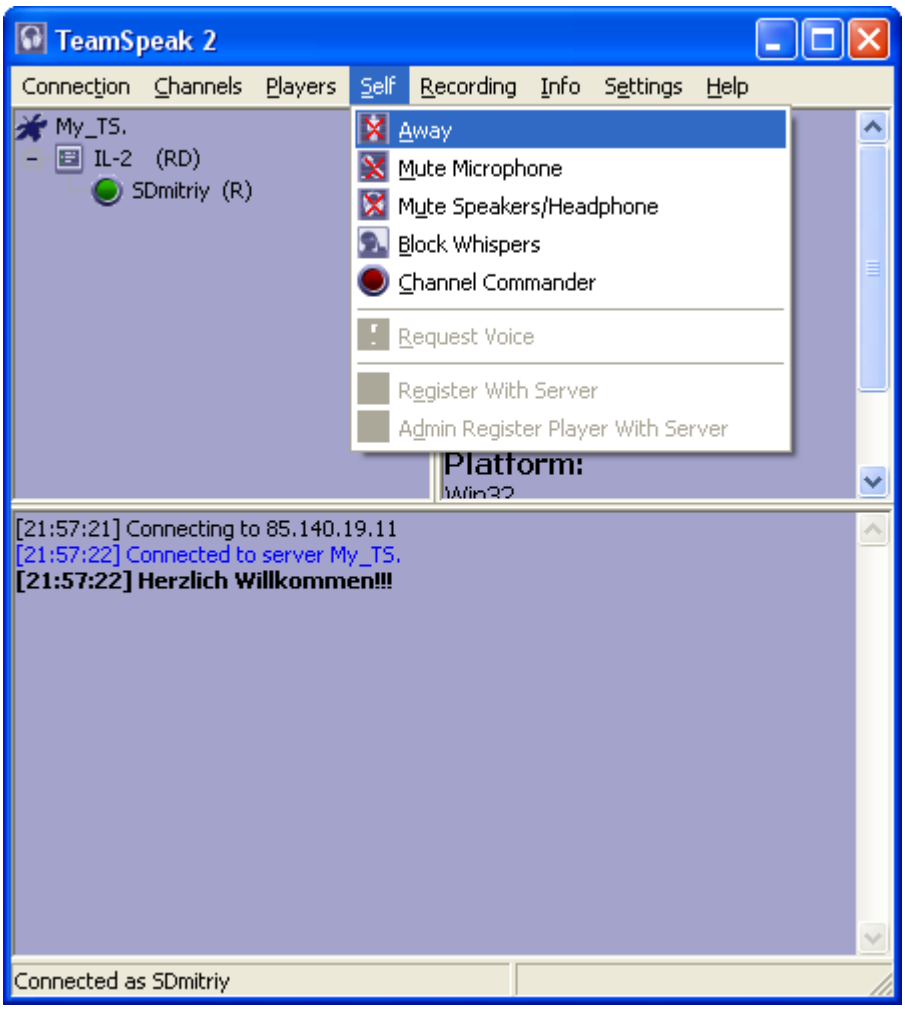


Рис. 82.

Для того, чтобы в будущем зайти на сервер, где вас зарегистрировали, со всеми своими полномочиями, надо проделать операцию над вкладкой **Local Adressbook (Локальная адресная книга)** в окне **Connection (Подключения)** -> **Connect (Соединиться)**. Если ваш сервер, еще не добавлен в **Local Adressbook (Локальная адресная книга)**, то его следует добавить вручную (не забывайте, что есть еще и возможность добавлять найденные серверы при помощи кнопки **Copy to Local adressbook (Скопировать в локальную адресную книгу)** на соседней вкладке **Web server list (Сервер лист)**).

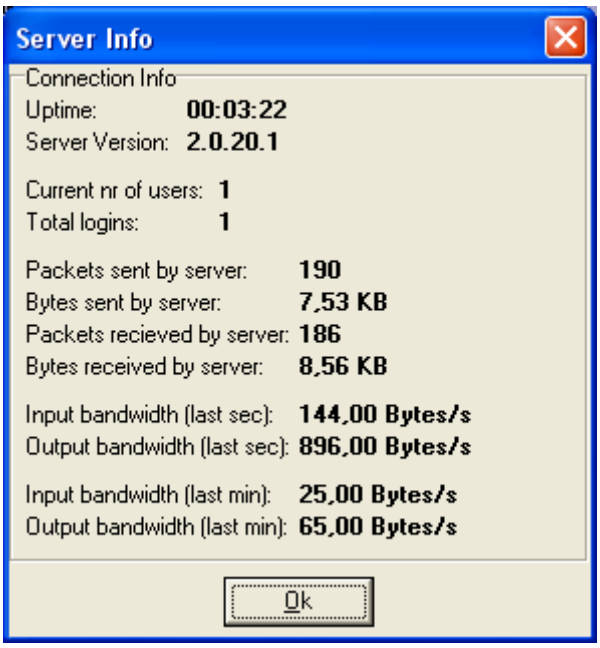


Рис. 83.

Параметр **Auto-Reconnect (Восстановить разорванную связь)** позволяет вам выбрать, будет ли автоматически произведена попытка восстановления связи с сервером **Teamspeak** в случае ее разрыва. Ниже находятся поля не обязательные к заполнению, а именно: **Default Channel (Стандартный канал)**, **Default Subchannel (Стандартный подканал)** и **Channel Password (Пароль на канале)**. Они позволяют вам сразу после установки соединения оказаться в указанном канале или подканале, если канал или подканал, указанный в поле **Default Channel (Стандартный канал)** или **Default Subchannel (Стандартный подканал)**, защищен паролем, то его надо предварительно де установки соединения указать в поле **Channel Password (Пароль на канале)**.

Описание серверной части TeamSpeak.

Сервер TeamSpeak (в дальнейшем просто TS или TC) можно загрузить с сайта http://www.goteamspeak.com/. Дистрибутив есть как под Windows, так и под Linux.

Точная ссылка на дистрибутив под **Windows**:
ftp://ftp.freenet.de/pub/4players/teamspeak.org/releases/ts2_server_rc2_20201.exe

При работе сервер использует порты **51234** и **14534** (**протокол TCP**) для администрирования и **8767 UDP** (по умолчанию) для хостинга.

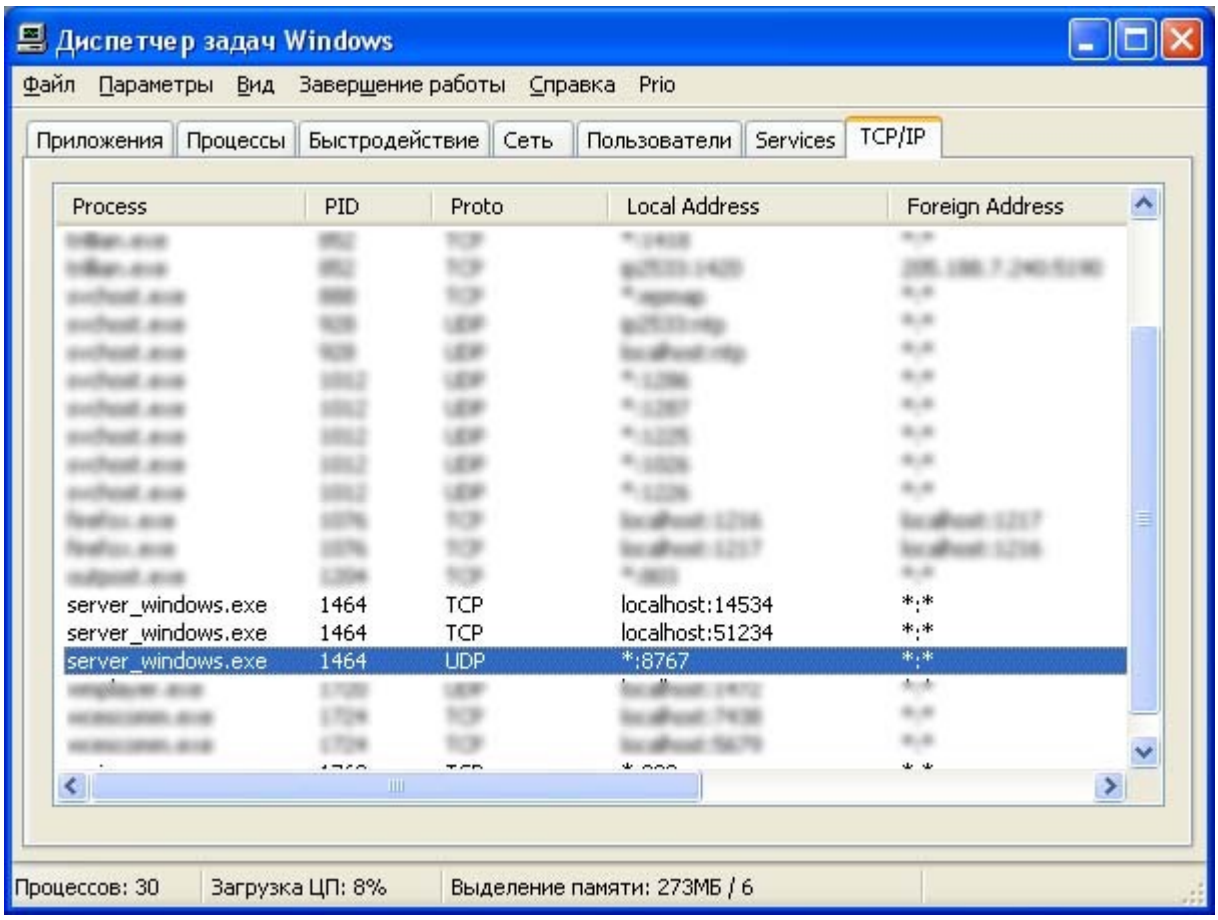


Рис. 84. **Использующиеся порты.**

Хочется заметить, что возможности администрирования сервера достаточно широки. По порту **51234** сервер можно администрировать с помощью **telnet**. А на порту **14534** расположен web-сервер администрирования. Рассмотрим сначала Web-сервер.

Для этого в адресной строке браузера введем адрес:
http://localhost:14534/

Перед нами появится окно регистрации пользователя. При установке сервера создаются два пользователя по умолчанию - **Admin** и **SuperAdmin**. Различие между ними достаточно большие**. Superadmin** имеет право создавать новый сервис TS (допустим на порту **8768**) плюс работа с сервисами. **Admin** может работать лишь с тем сервисом, на который его назначили. Назначение **admin**-а проводит Superadmin.

Рис. 85. **Окно входа для admin/registered user**

Это окно для входа пользователя admin или зарегистрированного пользователя. Как видно, здесь есть параметр **Serverport**. Это необходимо для того, чтобы администрировать разные сервисы **TS**, находящиеся на разных портах. Для **superadmin** этого параметра нет, так как он управляет всем сервером со всеми сервисами. Для входа под пользователем superadmin необходимо нажать на ссылку внизу.

Рис. 86. **Окно входа под пользователем superadmin**

Рассмотрим настройки доступные для superadmin, так как он имеет прав больше чем admin и следовательно больший доступ к настройкам.

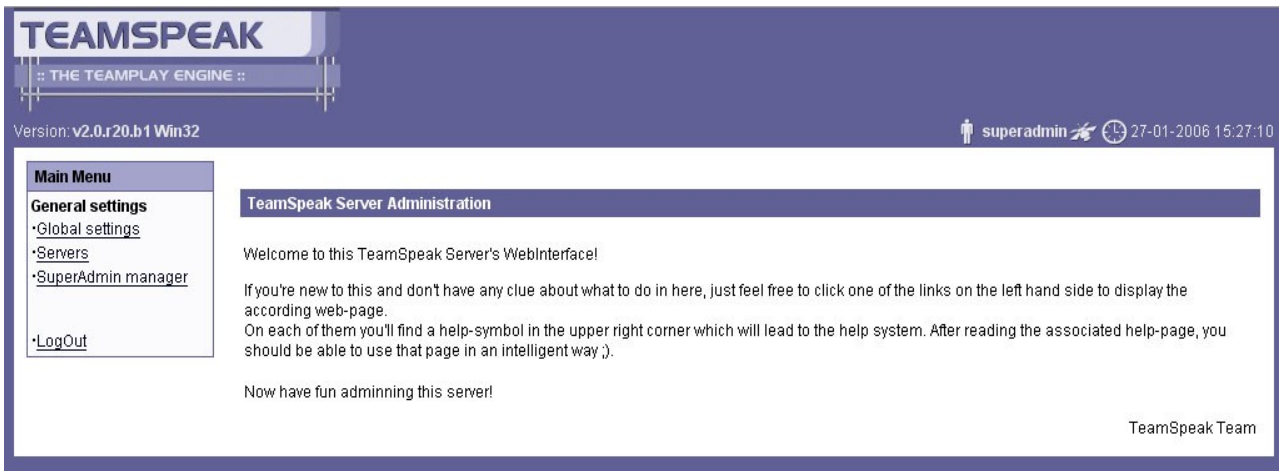


Рис. 87. **Главное меню управления для superadmin.**

После авторизации появляется главное меню администрирования сервера. Рассмотрим его подробнее.

Рис. 88.

Перед тем, как выбрать **Global settings,** нужно выбрать сервис, над которым вы собираетесь произвести настройку. В нашем случае сервер – один, по умолчанию можно сразу идти в **Global settings**.

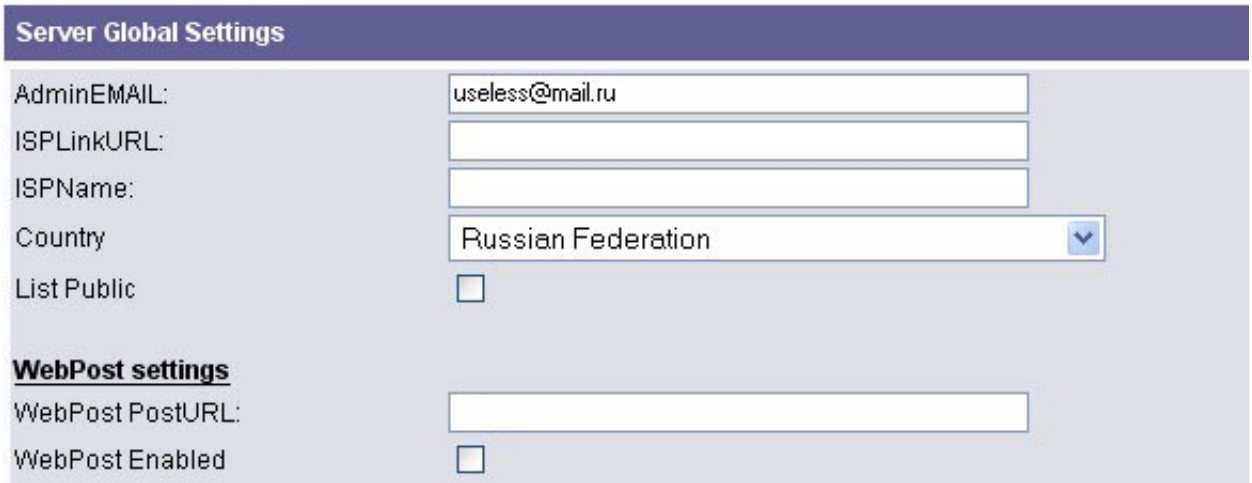


Рис. 89. **Окно Global settings.**

Здесь просто набивается информация о том как можно связаться с администратором сервера, провайдером, имя провайдера, страна, включать ли сервер в

список публичных серверов на **weblist.teamspeak.com**. Так же есть возможность выкладывать статистику и состояние сервера на сайт. Связь идет через **ms_sql**.

Рис. 90**. Окно выбора сервера.**

В этом окне осуществляется управление сервисами **TS** из рисунка видно, что мы используем один сервис на порту **8767**. По нажатию кнопок **stop, delete, select и add server** собственно и ведется работа над сервисами. При нажатии **Add server** открывается окно создания нового сервиса.

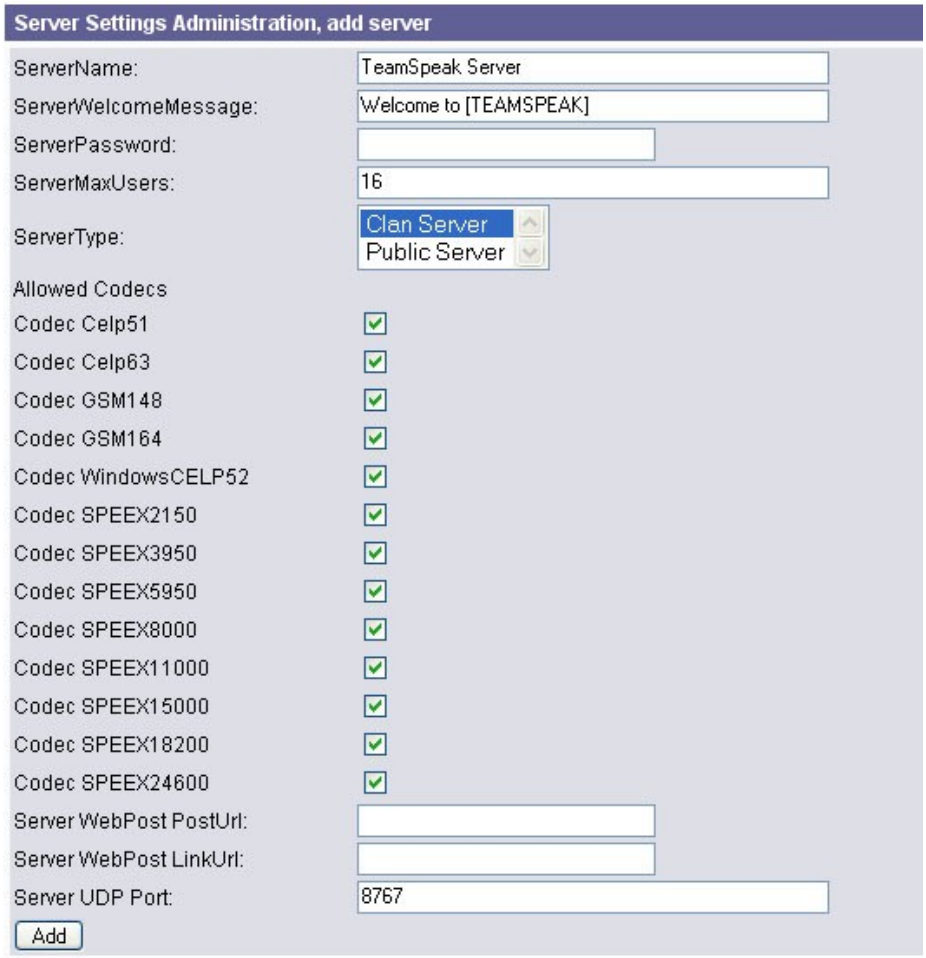


Рис. 91. **Создание нового сервера**

Здесь задается имя сервиса, сообщение при входе, пароль к серверу, максимальное число пользователей. Тип сервера: клановый или публичный. В случае выбора публичный, сервер будет добавлен в лист всех публичных серверов. Куча галок – это список кодеков при оцифровке речи, которые вообще можно использовать на сервере.

Celp - алгоритм оцифровки на основе предсказаний линейных изменений амплитуды голоса. Из достоинств - очень низкий расход трафика. Из недостатков – очень плохое качество. Кодеки **GSM** – оптимальные и по расходу трафика и по качеству звука. Обычно на канале ставим кодек **16,4 GSM.** Кодек Speex по качеству не уступает **GSM**, а с **18,200** обгоняет. Так что если трафика не жалко, а качество нужно максимальное-то Speex-ваш выбор при создании канала. Но об этом позже. Да, кстати, цифры рядом с кодеком указывают на расход трафика в **kbit/s.**

Выберем наш сервис (**Servers -> select server**). Изменилось и главное меню.

Рис. 92.

Добавились возможности управления выбранным сервером. О них подробнее.
Начнем с Server overview (просмотр описания сервера).

Server Settings Overview

General

ServerName:	**My_TS.**
ServerUDPPort:	**8767**
ServerId:	1
ServerVersion:	2.0.20.1
ServerPassword:	
ServerWelcomeMessage:	Herzlich Willkommen!!!
ServerWebPostLinkURL:	

Statistics

ServerUpTime:	**03:07:49**
CurrentChannels:	1
CurrentUsers:	3 / 16
ServerPacketsSend:	57732
ServerBytesSend:	7639821
ServerPacketsReceived:	50537
ServerBytesReceived:	5535187
Bandwitdh In [Bytes/sec]	40
Bandwitdh In [Bytes/min]:	56
Bandwitdh Out [Bytes/sec]:	48
Bandwitdh Out [Bytes/min]:	68

Рис. 93.

Как видим, чисто информационная страница. Приведено имя сервиса, порт, текущий номер (**id**) сервиса, версия, сообщение при входе и основная статистика. Настройка сервиса. Окно идентично настройкам нового сервиса.

Server Settings Administration

ServerName:	My_TS.
ServerWelcomeMessage:	Herzlich Willkommen!!!
ServerPassword:	
ServerMaxUsers:	16
Allowed Codecs	
Codec Celp51	☑
Codec Celp63	☑
Codec GSM148	☑
Codec GSM164	☑
Codec WindowsCELP52	☑
Codec SPEEX2150	☑
Codec SPEEX3950	☑
Codec SPEEX5950	☑
Codec SPEEX8000	☑
Codec SPEEX11000	☑
Codec SPEEX15000	☑
Codec SPEEX18200	☑
Codec SPEEX24600	☑
ServerType:	Clan Server / Public Server
Server WebPost PostUrl:	
Server WebPost LinkUrl:	
Server UDP Port:	8767

Save

Рис. 94.

В разделе Server Permissions рассматриваются права и возможности пользователей различного ранга.

Рис. 95.

Перейдем к Менеджеру пользователей.

Рис. 96.

Перед нами список зарегистрированных пользователей. Пользователей можно заносить в базу как здесь, нажав на кнопку **add client**, так и из клиента TS, если вы зашли под администратором или **superadmin-ом**. Создавать зарегистрированных пользователей вовсе не обязательно. Достаточно в настройках канала указать, что присоединятся могут и не зарегистрированные пользователи. Из этого меню можно назначить администратора сервера. Делается это так. Либо нажимаем кнопку edit около уже существующего юзера, и там ставим галку напротив server admin, либо при создании нового пользователя ставим галку напротив **server admin**.

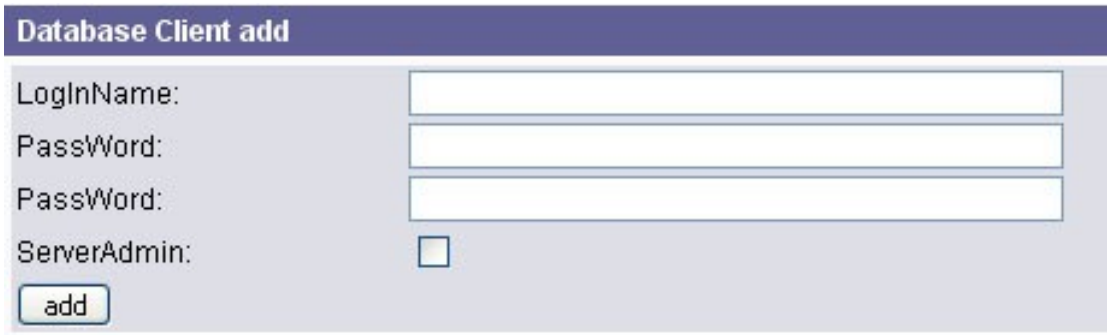


Рис. 97. **Взаимодействие сервера и клиента.**

При загрузке сервер занимает порт **8767 UDP** и начинает его “слушать” (все зависит от количества сервисов в настройках, данный порт - по умолчанию за первым сервисом). На сервер выделяется только один порт на обслуживание клиентов. Общение многих пользователей по одному порту достигается тем, что при передаче данных используется протокол **UDP**, не требующий предварительного подключения. В результате, сервер не выдает каждому клиенту по порту, а слушает только один, на который все клиенты посылают свою речь.

Клиент TS тоже имеет свой порт, который OS выдает динамически и соответственно IP адрес клиента. При получении датаграмм сервер запоминает, на каком порту и на каком IP сидит клиент. Приняв датаграмму **UDP** от клиента, сервер пересылает ее на все адреса, которые знает (по соответствующим портам клиентов), кроме того адреса, от которого он эту датаграму получил. Вот таким вот образом разговор одного человека слышат все остальные.

Технология общения:

При загрузке сервер занимает порт **8767 UDP** и начинает его “слушать” (все зависит от количества сервисов в настройках, данный порт - по умолчанию за первым сервисом). На сервер выделяется только один порт на обслуживание клиентов. Общение многих пользователей по одному порту достигается тем, что при передаче данных используется протокол **UDP**, не требующий предварительного подключения. В результате, сервер не выдает каждому клиенту по порту, а слушает только один, на который все клиенты посылают свою речь.

Клиент TS тоже имеет свой порт, который OS выдает динамически и соответственно IP адрес клиента. При получении датаграмм сервер запоминает, на каком порту и на каком IP находится клиент. Приняв датаграмму **UDP** от клиента, сервер пересылает ее на все адреса, которые знает (по соответствующим портам клиентов), кроме того адреса, от которого он эту датаграму получил. Благодаря этому разговор одного человека слышат все остальные.

Решение проблемы сообщения IP адреса сервера всем участникам общения.

Суть проблемы.

Как сообщить знакомым о том, что запущен сервер **TS** и что он находится на таком-то IP. Конечно, можно просто позвонить по телефону, сообщить по **Icq**, написать на **e-mail**.

Было бы легче, если бы был собственный web-сервер, тогда бы можно было просто с помощью скриптов, выполняемых сервером, и встроенных средств **TS server** передавать

статистику, настройки и т.д. Но у нас нет сервера, на котором мы могли бы выполнять те скрипты, исполнителем которых был бы сам сервер.

Имеется только учетная запись на каком-либо **ftp** сервере. Пусть это будет **front.ru**. Проблему можно решить написанием обыкновенного сценария **cmd**.

Решение.

Пошаговые операции на запуск.

1) Запустить сервер TS
2) Считать свой IP адрес, текущую дату и текущее время.
3) Поместить данные на сервер.
4) Проверить все ли в порядке.
5) Закрыть cmd.

Пошаговые операции на выход.

1) Удалить информацию с ftp сервера.
2) Закрыть TS server
3) Проверить правильность выполнения
4) Закрыть cmd.

Более подробно и по-порядку.

Сценарий на запуск.

1) Запустить сервер TS.
Команда "Z:\Program Files\Teamspeak2\Server\ server_windows.exe"

2) Считать свой IP адрес и локальное время.
Используются 3 команды.
time /t – ключ t для того, чтобы просто вывести время, а не запрос на ввод нового времени.
date /t – тоже что и для времени.
ipconfig – для того, чтобы узнать свой ip адрес.

Но есть небольшая сложность. А именно, мы должны куда-то эти данные сохранить. Кроме того, желательно чтобы была соблюдена html структура, так как файл будет выкладываться на сайт под именем index.html. Для того, чтобы начать вписывать данные в файл, файл нужно “подготовить”, точнее приготовить файл с расширением html и начинкой типа:

```
<html>
<head>
<title>My TS server</title>
</head>
<body>
Далее пойдет текст, возвращенный date /t.
<br>
Текст, возвращенный командой time /t
<br>
```

```
<br>
<pre>
Текст, возвращенный ipconfig-ом.
</pre>
</body>
</html>
```

Скрипт, который будет генерировать html файл с такой начинкой, будет выглядеть так:

```
@echo off
echo ^<html^>>g:\index.html
echo ^<head^>>>g:\index.html
echo ^<title^>>>g:\index.html
echo My TS server>>g:\index.html
echo ^</title^>>>g:\index.html
echo ^</head^>>>g:\index.html
echo ^<body^>>>g:\index.html
echo ^<center^>>>g:\index.html
echo Дата     >>g:\index.html
echo ^<B^>>>g:\index.html
echo ^<U^>>>g:\index.html
date /t>>g:\index.html
echo ^</U^>>>g:\index.html
echo ^</B^>>>g:\index.html
echo ^<br^>>>g:\index.html
echo Время запуска >>g:\index.html
echo ^<B^>>>g:\index.html
echo ^<U^>>>g:\index.html
time /t>>g:\index.html
echo ^</U^>>>g:\index.html
echo ^</B^>>>g:\index.html
echo ^</center^>>>g:\index.html
echo ^<br^>>>g:\index.html
echo ^<br^>>>g:\index.html
echo ^<PRE^>>>g:\index.html
ipconfig>>g:\index.html
echo ^</PRE^>>>g:\index.html
echo ^<br^>>>g:\index.html
echo ^<br^>>>g:\index.html
echo ^<br^>>>g:\index.html
echo ^<br^>>>g:\index.html
echo ^<br^>>>g:\index.html
echo ^<br^>>>g:\index.html
echo ^<br^>>>g:\index.html
echo ^</body^>>>g:\index.html
echo ^</html^>>>g:\index.html
@echo Generating html done.
@echo Sending index.html to server...
pause
cls
ftp -s:ftp_put_script.txt ftp.front.ru
@echo Sending done.
```

```
@echo Check output.
@echo deleting temporary index.html...
del index.html
@echo Done
Pause
```

Все понятно до команды ftp -s:ftp_put_script.txt ftp.front.ru

Эта команда запускает ftp клиент с указанием присоединится к хосту ftp.front.ru и при присоединении выполнить скрипт, написанный в файле ftp_put_script.txt. Листинг файла приложен здесь:

```
login
password
del index.html
lcd "g:\"
put index.html
bye
```

Где login-логин
Password – пароль
Дальше собственно команды.
1-удалить index.html если таковой имеется.
2-назначить локальным каталогом диск G:
3-загрузить на сервер index.html, находящийся на диске G.
4-Сказать серверу до свидания ☺.

В результате выполнения всех операций при заходе на страничку login.front.ru увидим следующее:

Рис. 98.

Ну и скрипт на завершение работы сервера.

```
@echo off
ftp -s:ftp_del_script.txt ftp.front.ru
@echo deleting done. Check output.
taskkill /im "server_windows.exe"
Pause
```

Немного подробнее. Файл ftp_del_script.txt содержит скрипт, исполняемый на ftp сервере ftp.front.ru. Листниг ниже.

```
login
password
del index.html
bye
```

Исполняется команда - del index.html, которая и удалит файл index.html с сервера.

Вот и решение данной проблемы. Достаточно чтобы человек заглянул на вашу страничку и он узнает, есть ли сервер TS, сколько он уже находится в uptime, и его ip адрес.

Приложение 2. Упрощённая методика создания многостраничного сайта.

Создать простой сайт – не сложно, создать хороший сайт – сложнее, создать профессиональный сайт – ещё сложнее. Но для создания любого сайта не нужно уметь программировать, достаточно умения работать с текстовым процессором.

Здесь рассматривается последовательность создания самого простого сайта, который можно сделать за несколько часов.

1. Создание одностраничного сайта.
2. Размещение сайта на Web-сервере.
3. Создание многостраничного сайта за час.

Рис. 99.

Создание одностраничного сайта.

1. С помощью текстового процессора Word создаём страницу, которую мы хотели бы видеть в качестве домашней страницы своего сайта в Internet. Называем её стандартным именем “index”.
2. Тематика создаваемой страницы произвольная. Она может быть посвящена лично Вам, учебному заведению или предприятию, на котором работаете Вы или Ваши близкие. Нужно только сделать страницу так, чтобы Вам не было за неё стыдно – она будет выставлена на всеобщее обозрение в Internet.
3. Сохраняем созданную страницу в формате doc, а затем – в формате htm (для этого через «файл -> сохранить, как» получаем на экране окно:
   в нижней средней части которого есть строка для определения типа файла. Выбираем надпись, у которой есть в скобках символы htm, и сохраняем полученную страницу. Получится имя страницы index.htm).

4. Если на странице содержались рисунки или другие элементы украшения, они будут помещены в графические файлы. Если таких графических файлов много, они помещаются в отдельную папку с тем же именем, но другим расширением.
5. Файл с расширением doc может понадобиться потом, если после размещения в Internet будет нужно откорректировать страницу.
6. При работе с объектами Internet все имена файлов и папок должны набираться латиницей, не содержать пробелов и иметь длину не более 8 символов. Саму страницу желательно создавать в корне жёсткого диска или в папке, находящейся в корневом директории и так же имеющей короткое имя.
7. После того, как страница создана и сохранена, необходимо переходить к размещению её в Internet. Для этого нужно зарегистрироваться на свободном сервере и перенести на него по ftp свои файлы и папки. Сайт создан.
8. Для проверки работоспособности сайта нужно в браузере набрать его имя (во время регистрации имя Вашего сайта выводится вместе с поздравлением) и нажать Enter.

Размещение сайта на Web-сервере

1. Регистрируем на «народе» свой сайт. Получаем login (имя сайта) и password (пароль). Например, login“zaitchik” с паролем 654321.
2. После регистрации автоматически попадаем в мастерскую, или заходим туда нажатием соответствующей кнопочки в меню.
3. Находим help, из которого извлекаем информацию о том, как зайти на сервер «народа» по ftp (Доступ к файлам по FTP).
4. В help`е обращаем внимание, как надо входить в ftp: ftp://логин:пароль@ftp.narod.ru (между логином и паролем стоит двоеточие).
5. Подставляем в эту формулу свои логин и пароль, получается что-то вроде: ftp://zaitchik:654321@ftp.narod.ru
6. С помощью браузера заходим по этому адресу, на экране появится дополнительное окно: Стрелка показывает отличительную особенность окна
7. Работать по ftp надо, как обычно в Windows. Можно создавать, удалять, копировать, перемещать файлы и папки, как на своём компьютере.

Создание многостраничного сайта за час.

1. В корне диска создать пустую папку с коротким именем, состоящим из цифр или латинских букв (лучше, если имя начинается с латинской буквы).
2. В созданной папке создать три Word-страницы: одну назвать «Index», вторую – 2, третью – 3.
3. Страницу «Index» заполнить такой информацией:

Страница «Index».

Страница 2.

Страница 3.

Рис. 100.

4. Вторую страницу заполнить такой информацией:

Страница 2.

Страница «Index».

Страница 3.

Рис. 101.

5. Третью страницу заполнить такой информацией:

Страница 3.

Страница «Index».

Страница 2.

Рис. 102.

6. Созданные страницы сохранить в виде Word-документа.
7. На созданных страницах некоторые поля объявляем гиперссылками. Для этого нужное поле выделяем, устанавливаем на него курсор, нажимаем правую кнопку мыши и выбираем функцию «гиперссылка» (если правая кнопка мыши отключена, после выделения нужного поля в основном меню Word выбираем пункт «Вставка – гиперссылка»):

В появившемся окне набираем имя файла, на который должен быть осуществлён переход по этой гиперссылке (файл 2.htm ещё пока не создан, но ссылку на него можно сделать).

После нажатия кнопки OK цвет гиперссылки изменится, а если на ней подержать кусор мыши, через несколько секунд появится адрес, по которому будет осуществляться переход при активизации этой гиперссылки:

Страница «Index».

Страница 2.

Страница 3.

Рис. 103.

Аналогично формируются: гиперссылка «Страница 3» на странице «Index»; страница «Index» и «Страница 3» на странице 2; страница «Index» и «Страница 2» на странице 3.

8. После формирования гиперссылок страницы сохраняются по команде «Формат – Сохранить как – Web-документ». В результате будут сформированы аналогичные страницы с расширением htm.
9. Активизируем страницу Index.htm и проверяем работоспособность гиперссылок. Если гиперссылки не работают, корректируем их, как указано в п.7.
10. Регистрируемся на сайте www.narod.ru или аналогичном ему. После регистрации заходим в Мастерскую и выбираем пункт «Работа с файлами по FTP». Далее начинается «Размещение сайта на Web-сервере».

Должно получиться что-то вроде: нажать (вернуться на сайт можно будет только с помощью кнопки «назад» в браузере).

Приложение 3. Обзор средств создания HELP - файлов

Microsoft разработала две программы Workshop для изготовления справочных систем. Речь идет о He1p Worcshop и **HTML He1p Workshop –** для файлов справки HLP - и HTML-форматов соответственно. Если бы работать с ними было удобно, то не

появлялись бы многочисленные утилиты, использующие вышеупомянутые средства лишь в качестве компиляторов. О таких утилитах и пойдет речь в этом обзоре.

Прежде всего – немного базовой информации о файлах справки в Windows. Наиболее популярны два вида Help - файлов: Windows He1p (*.HLP) и HTML He1p (* .CHM). Общаться с ними приходится через специальные API. Для Windows Help это ряд функций, описанных в winuser.h и импортированных из библиотеки user32. Функция WinHelp служит для управления приложением WinHe1p, передавая ему все необходимые параметры – имя файла помощи, контекст, макрокоманду (кроме того, макросы действуют также внутри HLP - файла), и так далее. Некоторые библиотеки предоставляют оболочки-надстройки для функции WinHelp, чтобы как - то скрыть использование громоздких макросов. Например, в Delphi класс TApplication имеет методы He1pJump, He1pCommand, и т.д.

Наиболее подробно о WinHelp'е повествует документ «Microsoft Windows Help Authoring Guide», который, если у вас нет диска Microsoft Developer's Library, можно скачать в Сети по адресу www.spnet.ru/neuro/help/whag.zip , а формат HLP есть в огромной БД по форматам – WOTSIT (www.wotsit.org) .

Рассмотрим теперь HTML He1p. Во избежание недоразумений отметим, что мы не затрагиваем в этой статье тему справочных систем, построенных из обычных HTML - файлов (как, например, это сделано в продуктах GIMP или Photoshop 6) – формат файлов HTML He1p ничего общего с ними не имеет.

Итак, «настоящий» HTML He1p - файл открывается при помощи приложения HTML He1p (HI.exe), который использует HTML He1p ActiveX Control (hhctrl.ocx). Сама же компонента hhctrl.ocx отвечает за функции навигации и управления окнами внутри He1p'a. Hhctrl.ocx может быть внедрена в справочную систему для обеспечения описанной функциональности (что и делается обычно по умолчанию).

К файлу формата HTML He1p обращаются с помощью функции HtmlHelp – примерно с теми же параметрами, что и для WinHe1p – это сделано для облегчения переноса обычных Help - систем на HTML He1p. HTML Help интегрирован с технологиями ActiveX, Java, скриптовыми языками, JScript и Visual Basic Scripting Edition (что подразумевает активное использование скриптов и апплетов – для навигации можно использовать, например, апплет HHCtrl.class), а также поддерживает изображения форматов JPEG, GIF и PNG.

Прежде чем рассматривать продукты, заметим, что для их нормальной работы (то есть доступа к He1p-компилятору) нужны He1p Workshop либо HTML Help Workshop – в зависимости от желаемого типа справочной системы.

He1pScribble 5.5. Производитель: Jan Goyvaerts. (www. Jgsoft.com)

Это не WYSIWYG [1]-редактор. Вы видите всю разметку документа непосредственно в формате Ruch Text; но благодаря многочисленным панелям инструментов создание He1p'a заметно ускоряется. В He1pScribble достаточно работать только с одним документом проекта – нет нужды создавать несколько файлов, как это делается в классическом Help Workshop. Интерфейс можно условно разделить на три важные части. С помощью плавающего слева списка навигация осуществляется по темам (topics), содержимое которых открывается в главном окне – удобном RTF - редакторе. А расположенные наверху панели инструментов позволяют проводить операции разметки – такие как вставка гиперссылок (внутри документа и для *WWW)*, выбор шрифтов, цветов, и так далее.

[1] "What you see is what you get" — средство разработки, в котором разрабатываемый продукт выглядит в точности так как его будет видеть пользователь.

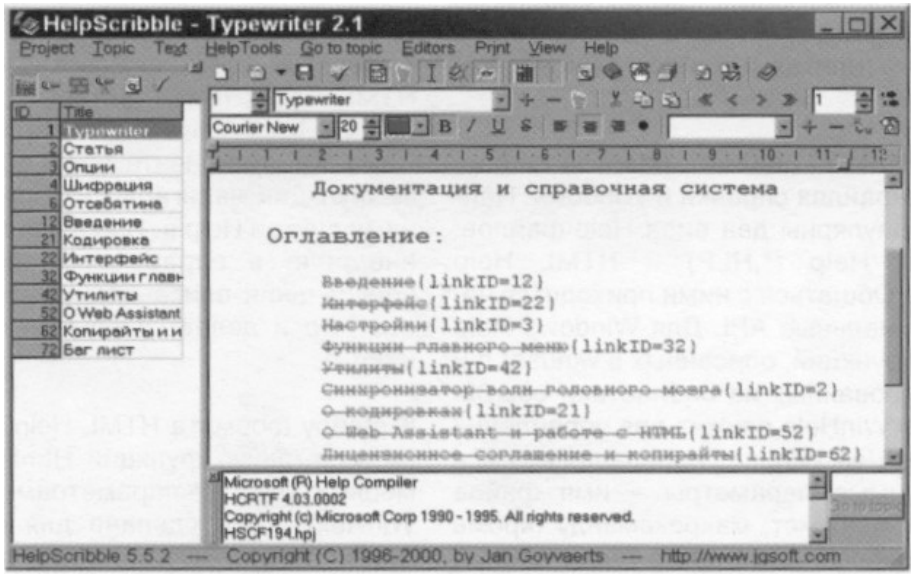


Рис. 104.

Из меню Editors доступны дополнительные редакторы, расширяющие возможности продукта. **SHG Editor** – средство для создания файлов формата Segmented HyperGraphics (SHG), которые являются примерными аналогами Image Maps в веб-дизайне. Изображение формата SHG разбито на области, каждая из которых может быть связана с некой темой в He1p-файле. Напомню, что HLP- файл может содержать в себе графику в форматах BMP, WMF/EMF и SHG. **Contents Editor –** редактор, составляющий Contents из топиков**. Windows Editor –** для редактирования характеристик главного и дополнительных (если таковые имеются) окон He1p-файла.

Кроме этих редакторов, He1pScribble предоставляет нам удобный редактор макросов для Winhelp, «мастера» вставки кнопок, картинок, утилиты конвертирования документа...

Вы можете экспортировать файл в формат HTML He1p, RTF и HPJ для работы с ними в Workshop'e, а также создать из справочной системы обыкновенный HTML-файл – при этом картинки будут сохранены в форматах PNG или GIF. Что до импорта, то вы можете импортировать файлы HPJ и RTF, что полезно при переносе проекта из Workshop'a.

Многочисленные опции позволяют откомпилировать документ не только как обычный He1p-файл для Win16 или Win32, но и сделать He1p для интеграции с He1p On1ine System в Delphi и Borland C++. Между прочим, интеграция самого HelpScribble с ними еще более близкая – инсталлировав в **1DE** пакет He1pContext Property Editor, вы получаете возможность, кликнув на свойстве He1pContext какого-нибудь компонента, автоматически вызвать редактор темы для этого компонента. Далее – «скормив» He1p Scribble какой-нибудь исходник модуля на Object Pascal, мы получим готовый каркас справочной системы для этого модуля.

Завершая рассказ о He1p Scribble, нельзя не заметить, что он распространяется совершенно бесплатно. Ограничения freeware-версии заключаются лишь в невозможности использования экспорта в HTML He1p, и вставкой в каждую страницу откомпилированного файла строки «This help file was created with He1p Scribble» мелким шрифтом.

Help & Manual 2.6.3. Производитель: EC Software. (www.ec-software.com/hmpage.htm)

Этот австрийский продукт – чистой воды WYSIWYG, хотя и похож внешне на He1p Scribble.

Во-первых, Help & Manual умеет создавать help'ы двух форматов обычный и HTML, а также экспортировать проект в виде HTML - документов.

Во-вторых, может строить темы на основе Delphi-проекта: «натравливаете» H&M на DRP-файл, его компоненты сканируются, и создается древовидный список тем или Table of contents (по желанию) из названий экземпляров классов, отображающий структуру приложения. В случае создания тем, каждой из них автоматически присваивается свой ID.

Редактирование текста происходит в двухстраничном окне с закладками, на одной из страниц которого представлен собственно текст в формате RTF, а на другой – свойства текущей темы. Выделяете слово, и по правой кнопке мыши появляется контекстное меню – хотите, делайте ссылку, а хотите – занесите выделенное слово в keywords одним легким нажатием кнопки. Очень быстро и удобно. Вставленные в текст листинги можно обработать встроенным средством Highlighter, которое поддерживает синтаксис Delphi, C++ Builder, Visual Basic и SQL.

С помощью Help & Manual вы можете оснастить свой He1p звуком и видео в формате AVI. Разумеется, поддерживается работа с HPJ-файлами. Мастер макросов, простейшее создание кнопок, полный контроль над внешним видом проекта, где настраиваются цвета, фреймы, панели навигации, и многое другое. Работу над большим проектом очень облегчает наличие в интерфейсе функции закладок и аннотаций. Для любителей повозиться с кодом существует возможность вставки в содержимое топика HTML-кода. И приятная новость напоследок: в H&M встроен декомпилятор HLP - файлов, который, однако, может не восстановить оригинальные ID'ы.

RoboHelp for Microsoft HTML He1p 2000. Производитель: eHe1p (Blue Sky Software). *(www.blue-sky.com)*

RoboHelp HTML генерирует справочные системы в трех форматах: HTML He1p, WeЬHe1p, и Sun Microsystems JavaHe1p (JAR), а импортировать умеет из форматов Windows Help (HLP), Windows Help Workshop (HPJ), HTML He1p Workshop (HHP) и plain HTML.

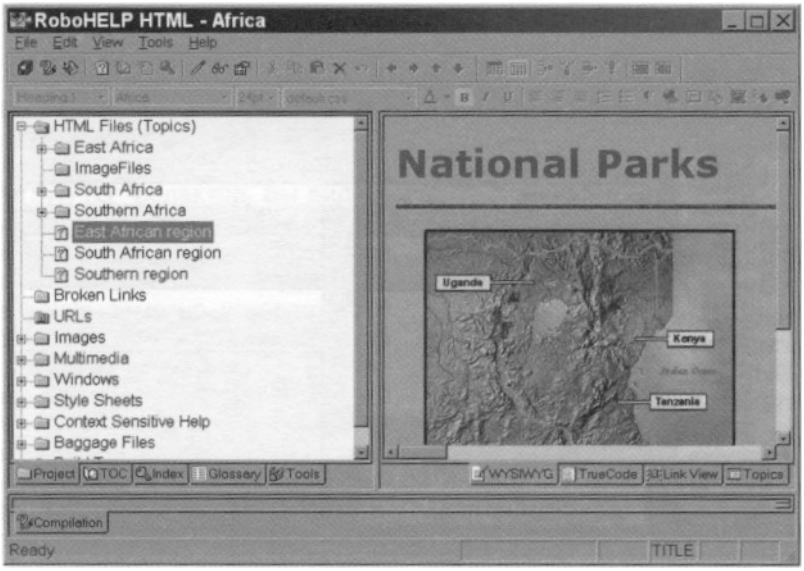


Рис. 105.

У продукта отличный интерфейс, концепция которого следующая: слева – многостраничный блокнот, где древовидно отображается структура проекта (темы в виде HTML -файлов, графические и мультимедийные ресурсы, style sheets, окна и т.д.). Table of Contents (TOC), индекс, глоссарий и утилиты. Справа – окно редактора, тоже в виде многостраничного блокнота. Можно использовать WYSIWYG, а можно перейти на страничку True Code и работать непосредственно с HTML. Страница Link View отображает схему структуры ссылок, использующихся в проекте, а, щелкнув мышью на закладке Topics («Темы»), можно вызвать список тем, содержимое которых открывается в виде листинга.

Пожалуй, по части менеджмента проекта RoboHelp обходит конкурентов – ведь, кроме продуманного интерфейса, он оснащен функцией Reports, которая выводит окно со статистикой избранных вами составляющих справочной системы в виде текста, который можно сохранить, скопировать или распечатать.

Как водится, имеются также «мастера» создания индекса и глоссария, а также необходимые в работе редакторы окон (и последовательности их просмотра), утилиты поиска тем и неправильных ссылок на них, удаления неиспользуемых Map ID'ов (это ID'ы, описанные в импортированных в проект заголовочных C/C++ *.h-файлах, где ID'ы определяются как "#define <ID_NAME> <ID_NUMBER>"), удаления неиспользуемых ключевых слов – и так далее. Словом, конечный документ будет «чистым», без залежей ненужного мусора.

Windows Help Designer 3.4. Производитель: Visage Software. (www.visagesoft.com)

Крайне эргономичный продукт, сочетающий в себе как WYSIWYG, так и RTF-редакторы, переключение между которыми происходит точь-в-точь, как в HomeSite – посредством страниц-закладок. Навигация по проекту осуществляется при помощи плавающих окон, которые при желании могут быть скрыты: это Contnts, Index/Keywords, и Topic Properties. Все интуитивно понятно. Например, если нужно создать кнопку и прописать для нее макрос – идем в Insert > Button, задаем надпись на кнопке, затем жмем «Add», и получаем окно ввода макроса, где также доступен мощный мастер, благодаря которому можно писать макросы, даже не зная макроязыка WinHelp. Так же очень хорош редактор таблиц, которые можно создавать прямо в тексте содержимого тем.

В меню Tools находится уникальную вещь – Delphi Parser. «Скармливаем» ей Delphi - исходники, и утилита создает на их основе заготовку для He1p'a, с отображением полной структуры иерархии классов, методов, переменных, процедур/функций, их синтаксиса и т.д. В результате получаем такой же Help, как в IDE Delphi – например, для класса создаются ссылки на Hierarchy и Members, а для каждого элемента указывается модуль, в котором он находится.

Другими словами, если вы написали библиотеку и хотите сделать к ней He1p, то достаточно "натравить" на исходники эту утилиту, и вам останется только вставить описания в сгенерированный каркас. В числе других интегрированных Tools – Screen Capture и простой Imege Editor для обработки изображений (функции рисования в нем, правда, отсутствуют).

He1p Delphi включает в себя мастера вставки изображений (GIF, JPEG ,BMP, SHG, WMF и EMF) и видеороликов AVI.

Взаимодействие с «внешней средой», поддерживается импорт ID'ов, описанных на C/C++, а экспортировать ID'ы можно для C/C++, VBasic и Pasca1. Для последнего, например, получаются файлы с записями вида: «Const CTX_Topic1=1;». Что до обычного импорта, то «понимается» RTF- формат. Родной же формат Windows He1p Designer –

обычные HPJ-файлы He1p Workshop'a (это к вопросу о совместимости). А «выходные» форматы таковы: WinпHe1p, JavaHe1p, Portable Document Format (PDF) и Plain HTML.

Help Magician 4.60. Прозводитель: Starline Software. *(www.helpmagician.com)*

Пятая версия пока что только обещается разработчиками; пользоваться они предлагают старой, четвертой. Это довольно громоздкий (по сравнению с He1p Magician 5) WUSIWYG - генератор He1p'ов с интерфейсом на любителя. Умеет делать файлы Windows He1p и HTML He1p.

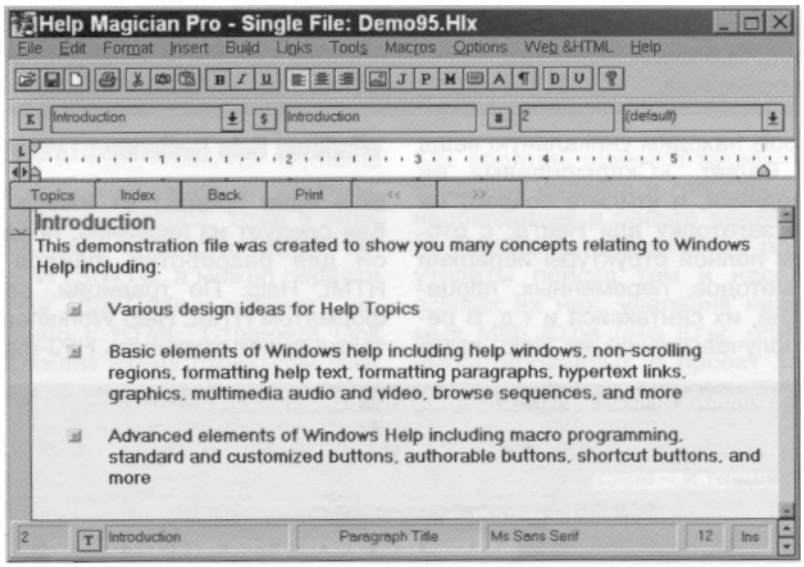


Рис. 106.

Продукт очень сильно «погряз» своими корнями в Win16, что видно не только по интерфейсу, но и по поддержке древних форматов вроде Word 1.0. Тех пользователей, которых не отпугнет интерфейс, обрадует громадное количество всевозможных опций и мастеров. Рассмотрим наиболее полезные из возможностей продукта.

VBasic He1p Wizard делает He1p - оболочку для VB - проекта, сканируя его код на предмет обнаружения свойств He1pContextID у объектов. Система закладок позволяет спокойно разрабатывать большие проекты. Редактор Shed поможет создать из BMP- или WMF-файла Image-Map формата *.SHG, либо позволит редактировать уже существующий *.SHG.

Кстати о мультимедиа – кроме вставки изображений многих форматов (BMP, WMF, SHG, ICO, PCX и TGA), в He1p Magician можно использовать файлы WAV, MID и внедрять AVI (все это добро проигрывается через MCI). Подобно Word'у, продукт позволяет создавать шрифтовые стили. Еще одним хорошим аспектом является поддержка групповой разработки проекта.

Приложение 4. Техника работы с мастером Windows Application.

При входе в Microsoft Visual Studio .NET выбрать Windows Application, имя проекта и определить папку для хранения проекта.

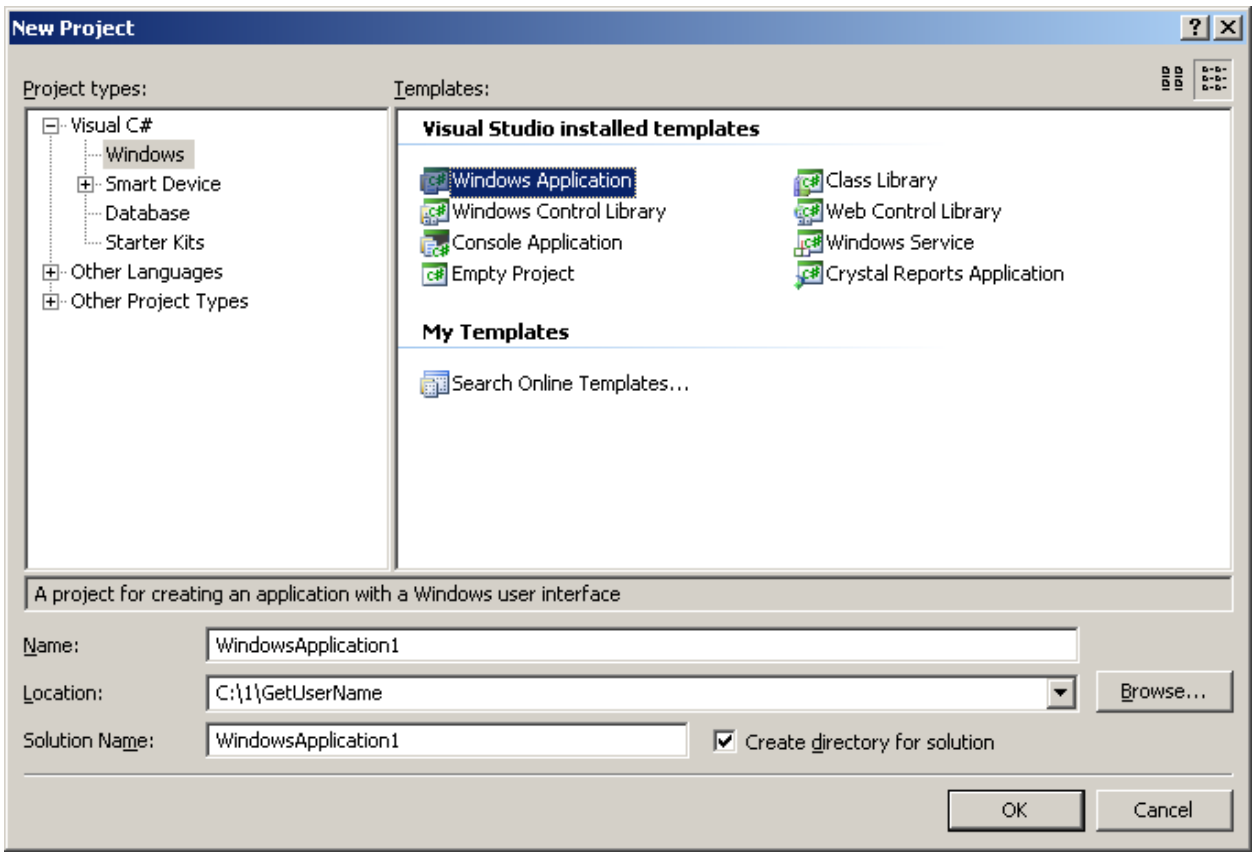


Рис. 107.

Открывается окно Microsoft Visual Studio .NET, в котором находится окно формы:

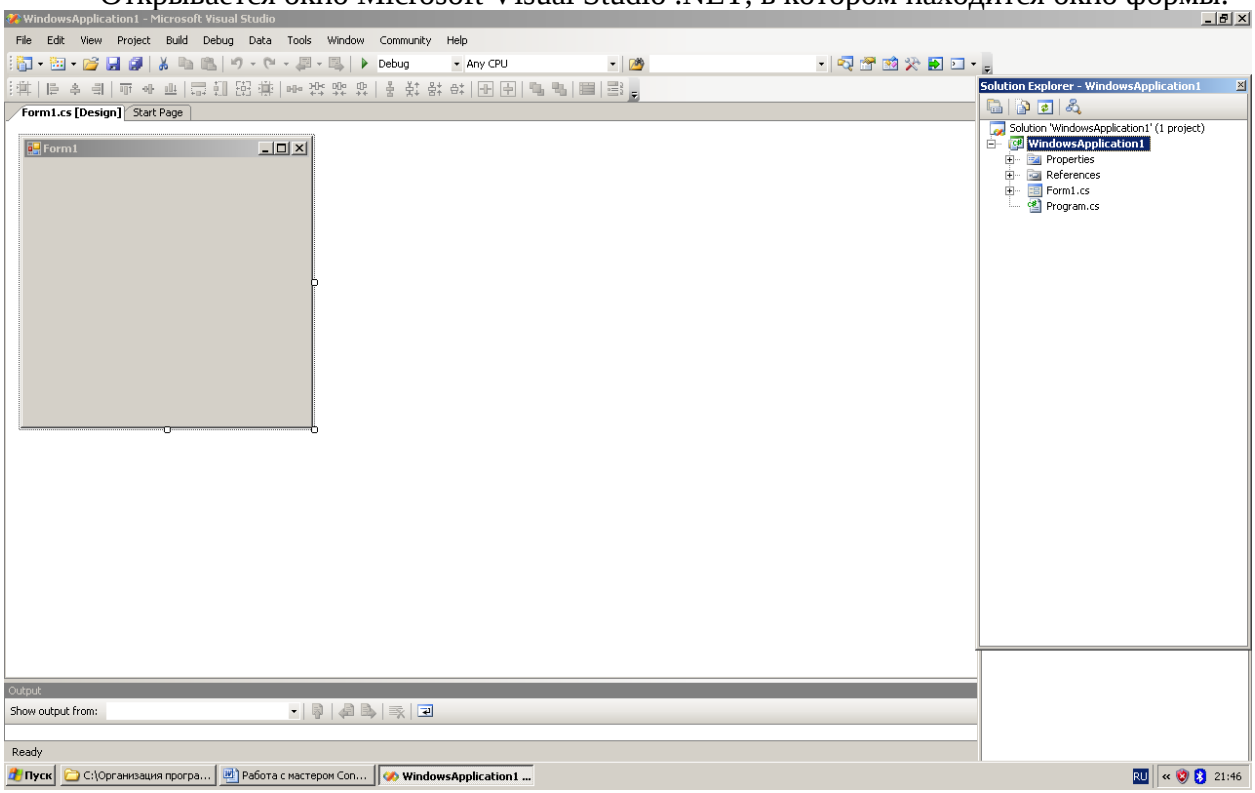


Рис. 108.

Форма предназначена для реализации интерфейса пользователя в приложении. Содержит большой набор свойств, методов, событий для реализации различных вариантов пользовательского интерфейса. Является окном и наследует классу Control.

Форма — это класс. Экземпляр этого класса: объект-представитель класса Form — это окно, которое появляется в Windows- приложении.

Этот класс можно использовать как основу для создания различных вариантов окошек:

- стандартных;
- инструментальных;
- всплывающих;
- borderless;
- диалоговых.

Создается "окно", в котором можно разместить элементы управления, обеспечивающие интерактивное взаимодействие приложения и пользователя (например, «заполните форму, please»).

Известна особая категория форм – формы MDI – формы с многодокументным интерфейсом (the multiple document interface).

Эти формы могут содержать другие формы, которые в этом случае называются MDI child forms. MDI-форма создается после установки в true свойства IsMdiContainer.

Используя доступные в классе Формы свойства, можно определять внешний вид, размер, цвет и особенности управления создаваемого диалога:

- свойство Text позволяет специфицировать надпись в заголовке окна приложения;
- свойства Size и DesktopLocation позволяют определять размеры и положение окна в момент его появления на экране монитора;
- свойство ForeColor позволяет изменить предопределенный foreground цвет всех расположенных на форме элементов управления;
- свойства FormBorderStyle, MinimizeBox и MaximizeBox позволяют изменять размеры окна приложения во время его выполнения.

Методы класса обеспечивают управление формой:

- например, метод ShowDialog обеспечивает представление формы как модального диалога;
- метод Show показывает форму как немодальный диалог;
- метод SetDesktopLocation используется для позиционирования формы на поверхности Рабочего стола.

Форма может использоваться как стартовый класс приложения. При этом класс формы должен содержать точку входа – статический метод Main. В теле этого метода обычно размещается код, обеспечивающий создание и формирование внешнего вида формы.

Обычно заготовка формы "пишется" мастером.

При инициализации формы (реализации метода InitializeComponent()) мастер создаёт файл Form1.Designer.cs. Это зона ответственности мастера приложений. Категорически не рекомендуется в этом методе делать что-либо самостоятельно — во избежание потери всего того, что там может быть построено, поскольку Мастер приложения может изменить содержимое тела метода в соответствии с изменениями внешнего вида приложения.

Важна строчка в теле функции Main:
Application.Run(new Form1());

В принципе, эта строчка и отвечает за создание, "запуск" в потоке приложения и возможное появление на экране дисплея формы.

Управление жизненным циклом и отображением форм и всего, что на них находится осуществляется следующими методами:

```
Form.Show(),
Form.ShowDialog(),
Form.Activate(),
Form.Hide(),
Form.Close().
```

Имеет смысл рассмотреть **события**, связанные с созданием, функционированием и уничтожением формы. При щелчке по пустому пространству на форме в правом углу окна появляется панель Properties. На эту панель можно вывести либо свойства, либо события – переключаются эти возможности значками: - показать свойства:

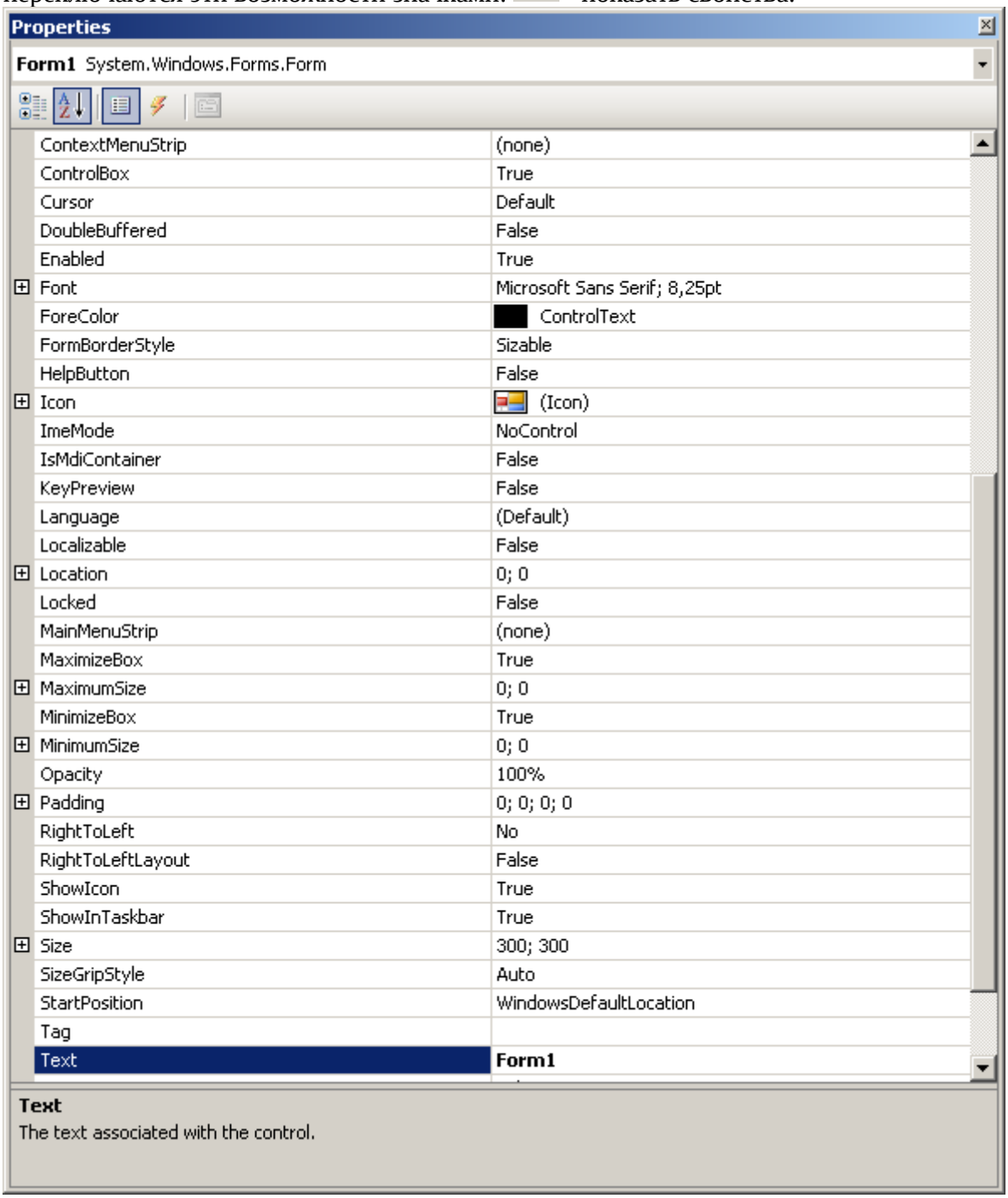


Рис. 109.

и - показать события:

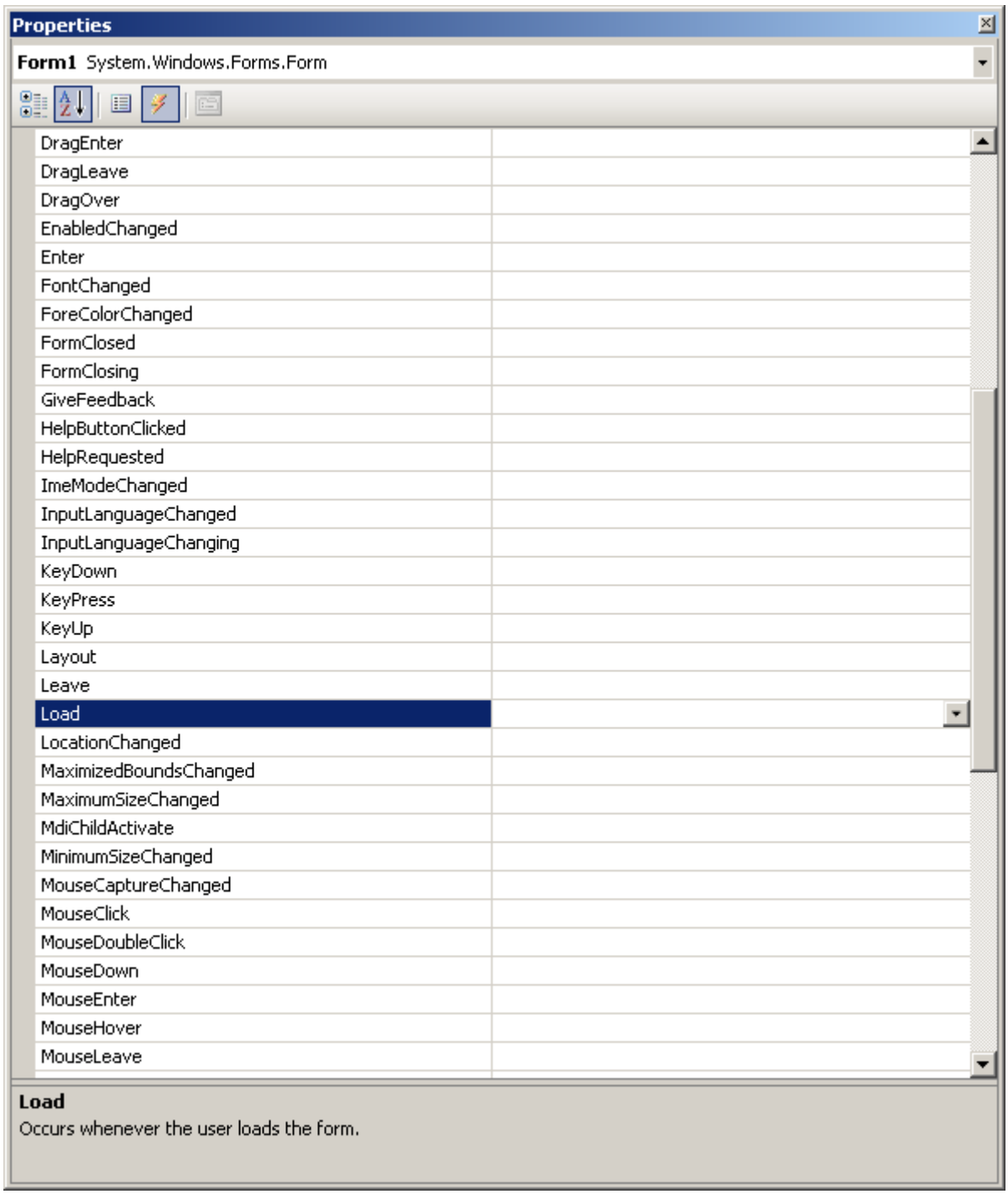


Рис. 110.

К числу событий относятся:

- Load. Генерируется ОДИН РАЗ, непосредственно после первого вызова метода Form.Show() или Form.ShowDialog(). Это событие можно использовать для первоначальной инициализации переменных и для подготовки формы к работе. Сколько в приложении форм, столько раз будет генерироваться это событие. Назначение максимальных и минимальных размеров формы – для этого более подходящего места, нежели обработчик события OnLoad, не найти!
- Activated. Многократно генерируется в течение жизни формы, когда Windows активизирует данную форму. Связано с получением и потерей фокуса. Все необходимые мероприятия выполняются здесь. Методы Form.Show(), Form.ShowDialog(), Form.Activate() (передача фокуса, реализованная программно!) способствуют этому. Передача фокуса

элементу управления (кнопка – это тоже окно) сопровождается автоматическим изменением цвета элемента управления.

- VisibleChanged. Генерируется всякий раз при изменении свойства Visible-формы — когда она становится видимой или невидимой. Событию способствуют методы Form.Show(), Form.ShowDialog(), Form.Hide(), Form.Close().
- Deactivated. Возникает при потере фокуса формой в результате взаимодействия с пользовательским интерфейсом либо в результате вызова методов Form.Hide() или Form.Close() – но только для активной формы. Если закрывать неактивную форму, событие не произойдет! Activated и Deactivated возбуждаются только при перемещении фокуса в пределах приложения. При переключении с одного приложения на другое эти события не генерируются.
- Closing. Непосредственно перед закрытием формы. В этот момент процесс закрытия формы может быть приостановлен и вообще отменен, чему способствует размещаемый в теле обработчика события следующий программный код: e.Cancel = true; // e – событие типа CancelEventArgs.
- Closed. Уже после закрытия формы. Назад пути нет. В обработчике этого события размещается любой код для "очистки" (освобождения ресурсов) после закрытия формы.

Получив на экране окно с формой, через View - > ToolBox вызываем в окно панель инструментов:

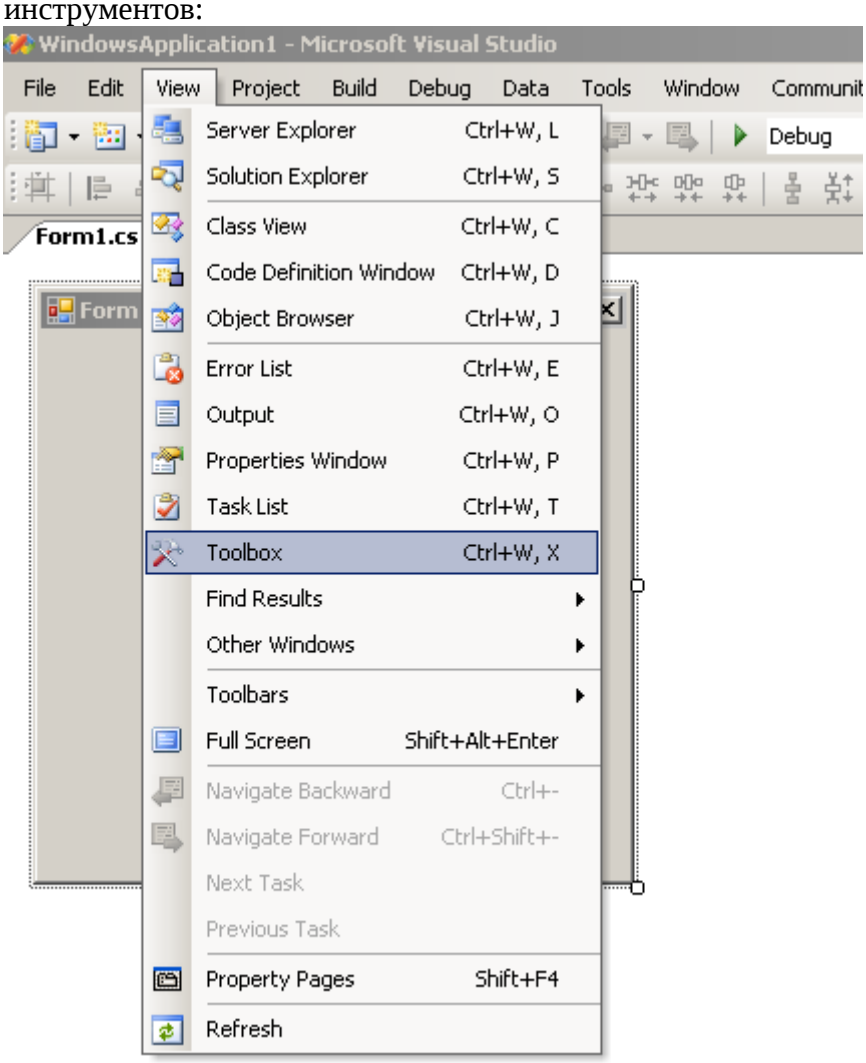


Рис. 111.

В основном окне пакета появляется панель инструментов:

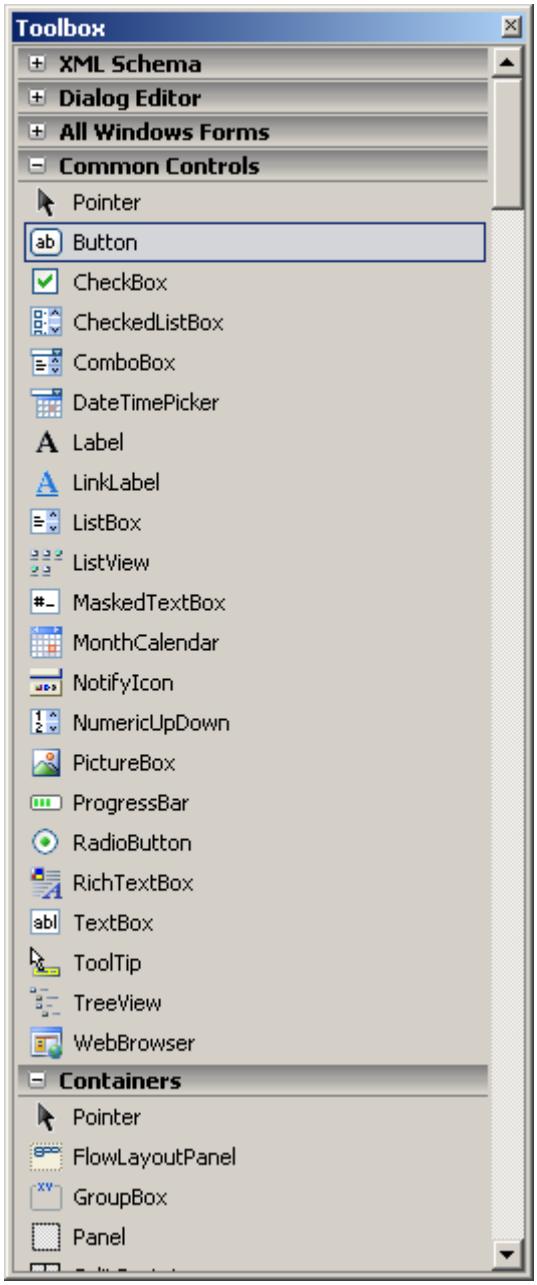


Рис. 112.

На панели инструментов (её свободно можно перемещать относительно окна мышкой, зацепив панель за синюю полосу с надписью Toolbox) выбираем кнопку (Button) и мышкой переносим её на форму:

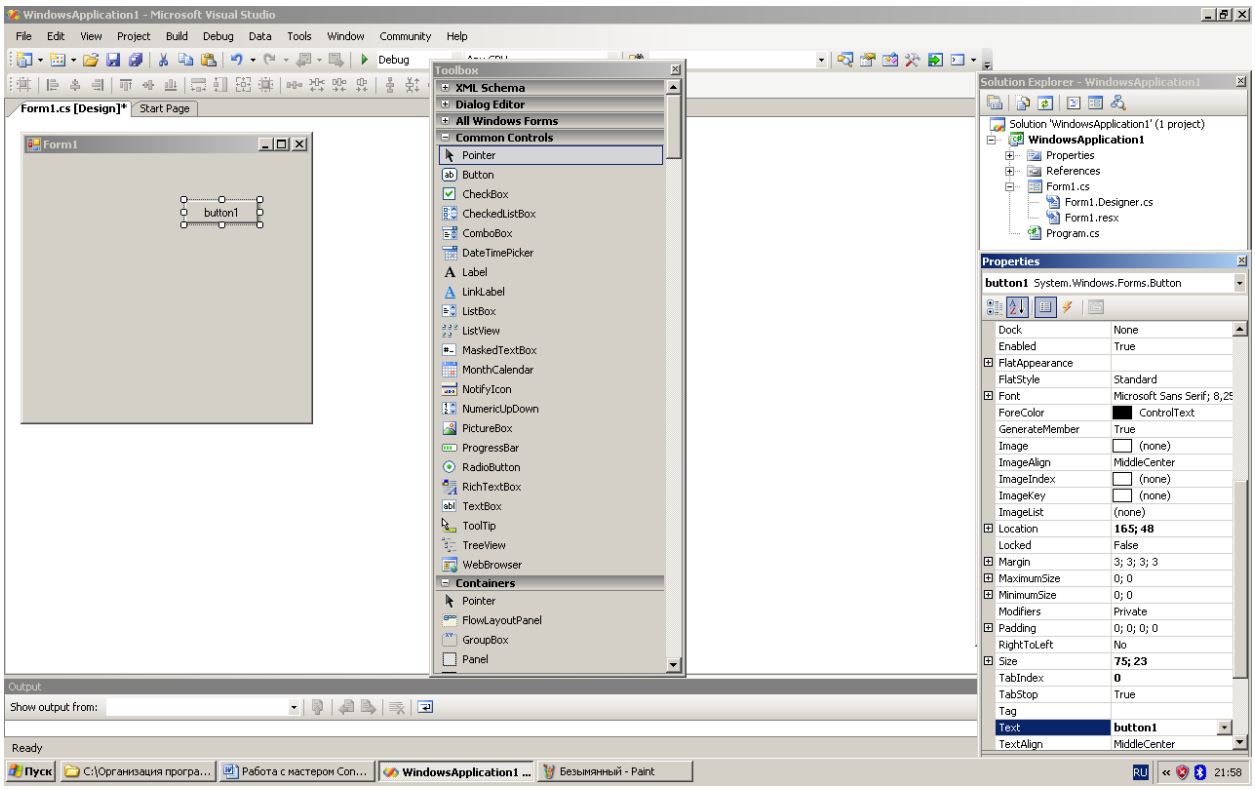


Рис. 113.

Кнопка (Button) выделена, и в правой части окна появляется панель её свойств, которую так же можно перемещать по экрану:

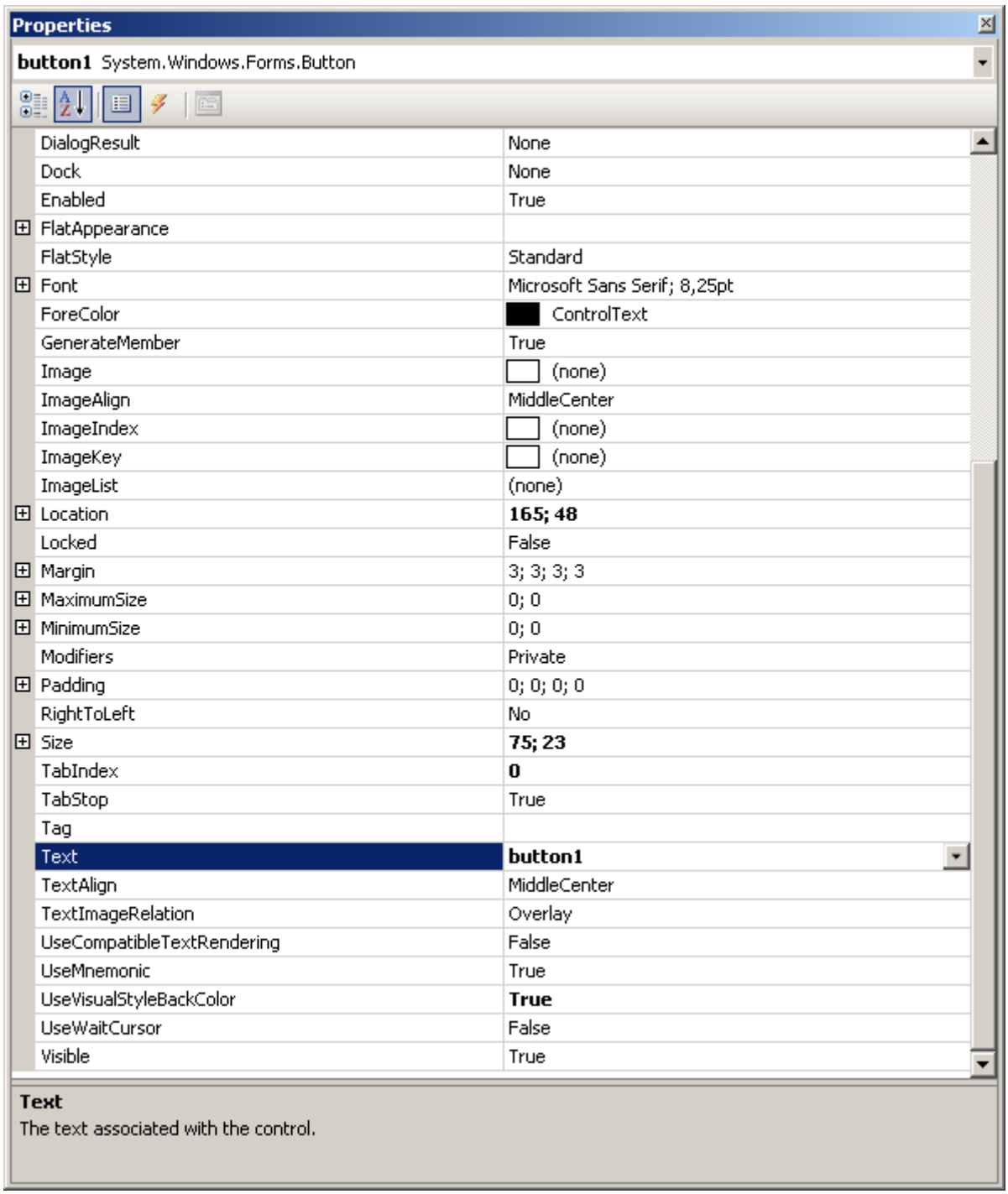


Рис. 114.

Свойство Text хранит надпись на кнопке, обычно связанную с тем, что произойдёт, если на кнопку нажать. Установим надпись: «Нажать». Кроме того, активируем событие Click: запишем значение этого события button1_Click. Мастер выведет следующее содержание файла Form1.cs:

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Text;
7  using System.Windows.Forms;
8
9  namespace WindowsApplication1
10 {
11     public partial class Form1 : Form
12     {
13         public Form1()
14         {
15             InitializeComponent();
16         }
17
18         private void button1_Click(object sender, EventArgs e)
19         {
20
21         }
22     }
23 }
```

Рис.

На рис. видно место для программы, которая будет исполняться при нажатии на кнопку (строки 18 – 21). Запишем на строке 20 команду

```
MessageBox.Show("Здравствуйте");
```

откомпилируем и выполним программу:

Рис. 115.

При нажатии кнопки появляется окно:

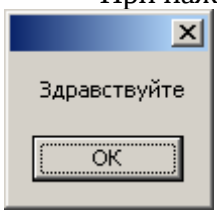


Рис. 116.

В результате проделанной работы мастером формируется проект, в состав которого

входят:

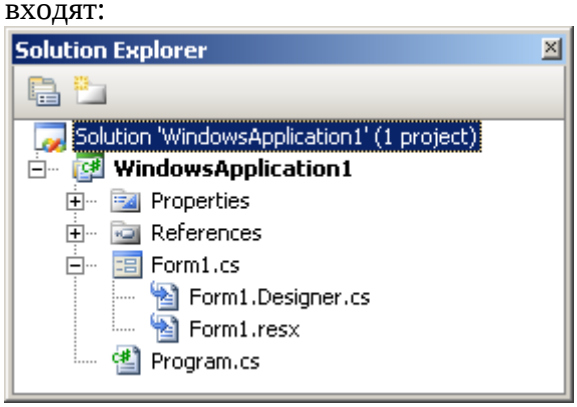


Рис. 117.

Сетевое электронное издание монографии.

Кириченко А.А.
профессор департамента «Программная инженерия» факультета компьютерных наук Федерального государственного автономного образовательного учреждения высшего профессионального образования “Национальный исследовательский университет "Высшая школа экономики" при правительстве РФ”.

«Технологии экстремального программирования»

Рецензент: к.т.н., доцент Гудыно Л.П.

Редактор Альбицкая Н.Б.

ISBN 978-5-9904911-3-7

9 785990 491137