

**Принципы разработки
программного обеспечения
с учетом необходимости
его повторного использования,
сопровождения и модификации и их
поддержка в среде разработки
IBM Rational Rhapsody**

Дмитрий Рыжов

**Руководитель
направления IBM Rational
SWD Software**



Содержание

- Текущая ситуации при разработке ПО
- Эволюция подходов по разработке ПО
- Возможности IBM Rational Rhapsody по разработке ПО

Типичная ситуация в проекте по разработке ПО

- Увеличивающаяся сложность ПО
- Трудности повторного использования наработок в линейке продуктов
 - Немодульная разработка приложений очень усложняет повторное использование кода в продуктовых линейках
- Постоянная нехватка времени
- Трудности понимания кода в отсутствии проектной документации
 - Часто лучшая или единственная документация - это сам код
 - Разработчик кода часто уже не работает в компании
 - Новые разработчики тратят кучу времени на изучение имеющегося кода
 - Повторное использование очень сложно из-за отсутствия понимания имеющегося кода
- Тестирование производится на последних этапах разработки, приводя к обнаружению ошибок, когда их наиболее дорого исправлять
 - Функциональность продукта урезается
 - Продукты отзываются

Принципы разработки, важные для повторного использования, сопровождения и доработки

- Необходимость проектирования
 - Модульность
 - Четко проработанные интерфейсы и протоколы взаимодействия
- Необходимость качественной актуальной документации
- Отделение прикладной логики от реализации на платформе
 - Сохранение прикладных наработок при изменении технологической платформы
- Наличие тестов для регрессионного тестирования при дальнейших изменениях
- Управление конфигурациями

Эволюция подходов к разработке ПО

- Программный код – единственная документация
- Документирование в текстовом редакторе
- Документирование на основе моделей
- Документирование в коде

- Разработка на основе моделей (MDD)
- Тестирование на основе моделей (MDT)
- Архитектура на основе моделей (MDA)

Программный код – единственная документация

- Часто является единственным выходом процесса разработки
- Артефакты проектирования остались в головах и на салфетках
- Прикладная логика перемешана с вопросами ее реализации
- Чужой код – не есть свой код
- Как следствие очень трудно разобраться
- А значит трудно повторно использовать, поддерживать, модифицировать
- Легче переписать, что постоянное и делается

Документирование в текстовом редакторе

- Документирование создает дополнительную нагрузку на разработчиков
- Если и ведется, то требует очень больших затрат
- Трудно сделать согласованной и непротиворечивой, так ее невозможно проверить
- Трудно синхронизировать с кодом
- Часто устаревает и в результате выбрасывается

Документирование на основе моделей

- Создание моделей на языке UML давно и активно используется при разработке ПО
- UML стал стандартом де факто при проектировании ПО
- Эффективно для создания согласованного описания ПО
- Позволяет автоматически генерировать документацию на основе моделей

Недостатки:

- Модели не синхронизированы с кодом
- Документация на основе моделей устаревает и выбрасывается также как обычная документация
- Создает дополнительную нагрузку на разработчиков ПО

Документирование в коде

- Совмещение документации и кода в одном источнике (literal programming)
- Центральным артефактом является программный код
- Генерация документации на основе кода (doxygen и т.д.)
- Автоматический анализ отношений в коде и генерация структурных диаграмм на языке UML

Недостатки

- Программный код не подходит для проектирования
- Артефакты проектирования все так же остаются в головах и на салфетках
- Только структурная документация и описание интерфейсов (API)
- Документация не содержит описания поведения

Разработка ПО на основе моделей (MDD)

- Объединение лучших практик
- Центральным артефактом разработки является модель
- Модель – это объединение информации в одном источнике: проектирования, программирования и документирования
- Модель может редактироваться как на уровне языка моделирования, так и кода
- Код и документация автоматически генерируется из модели путем ее трансформации

Преимущества:

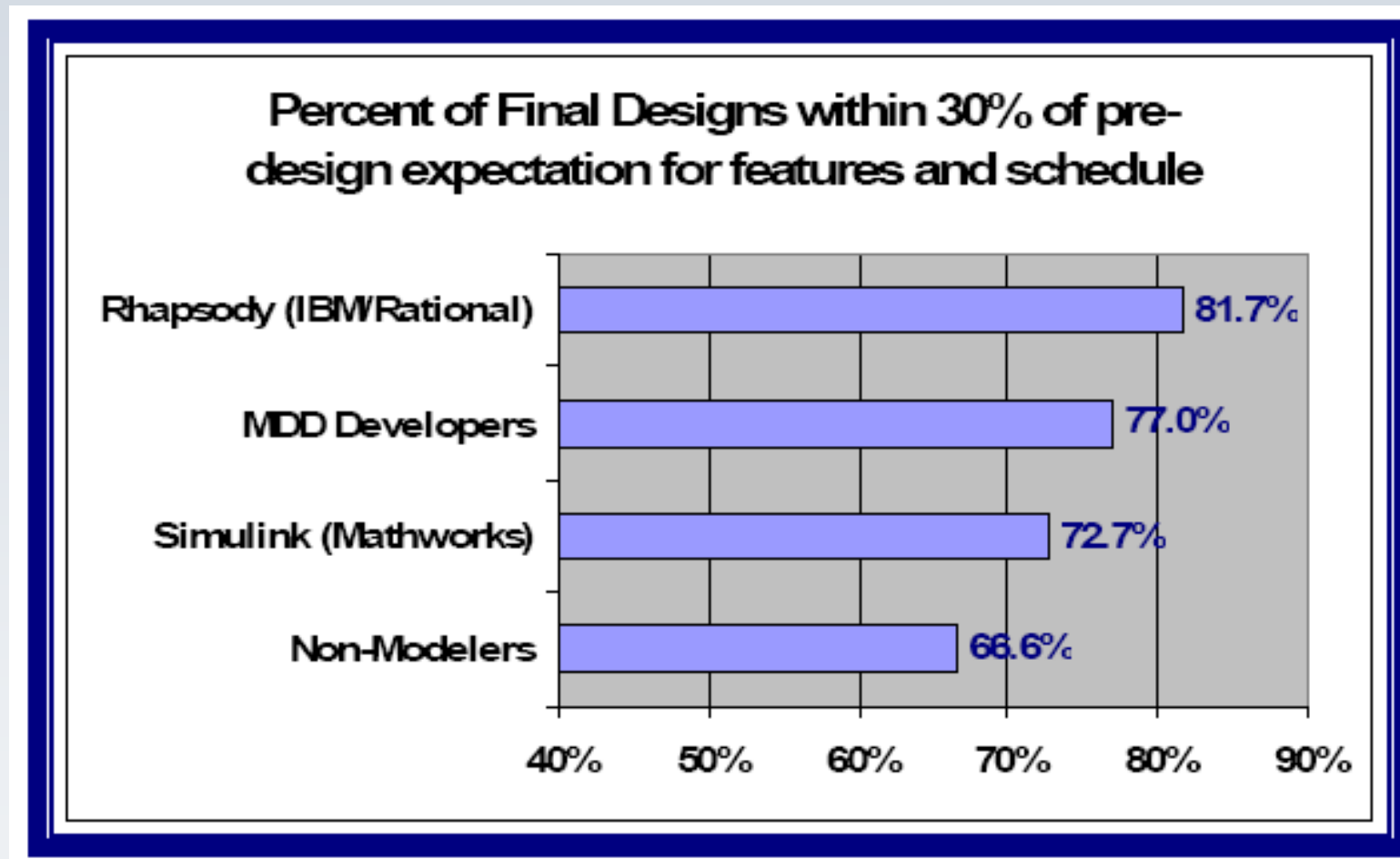
- Модель и код всегда синхронизированными друг с другом
- Проектирование становится частью процесса разработки
- Документирование и проектирование больше не есть дополнительная нагрузка

Архитектура на основе моделей (MDA)

- Следующий логический шаг от MDD
- Многоуровневое моделирование
- Различные уровни описывают решение с различной детализацией
- Пример трехуровневого подхода
 - модель анализа
 - модель проектирования
 - модель реализации
- На каждом уровне применяются свои паттерны проектирования
- Между уровнями применяются свои трансформации
- Многие трансформации могут быть автоматизированы

Разработчики рапортуяют...

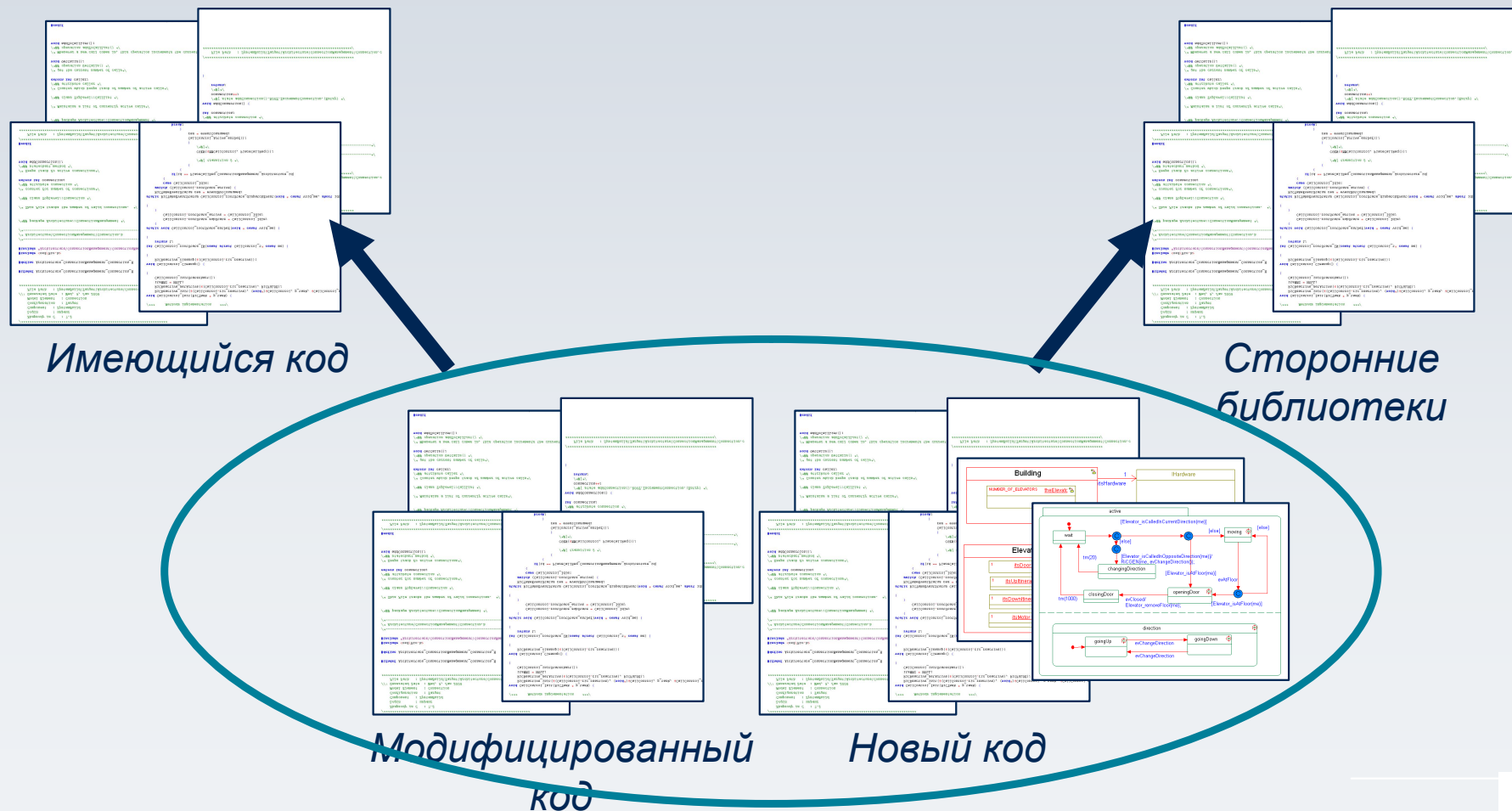
- MDD более эффективно, чем традиционное ручное кодирование!



Источник: *A Model Driven Approach to Software Development for Systems*, Embedded Market Forecasters, Nov. 2008

Типичный проект по разработке ПО

В типичном проекте по разработке ПО создается новый код, делаются изменения в имеющемся коде, часть имеющегося кода используется без изменений, подключаются сторонние библиотеки

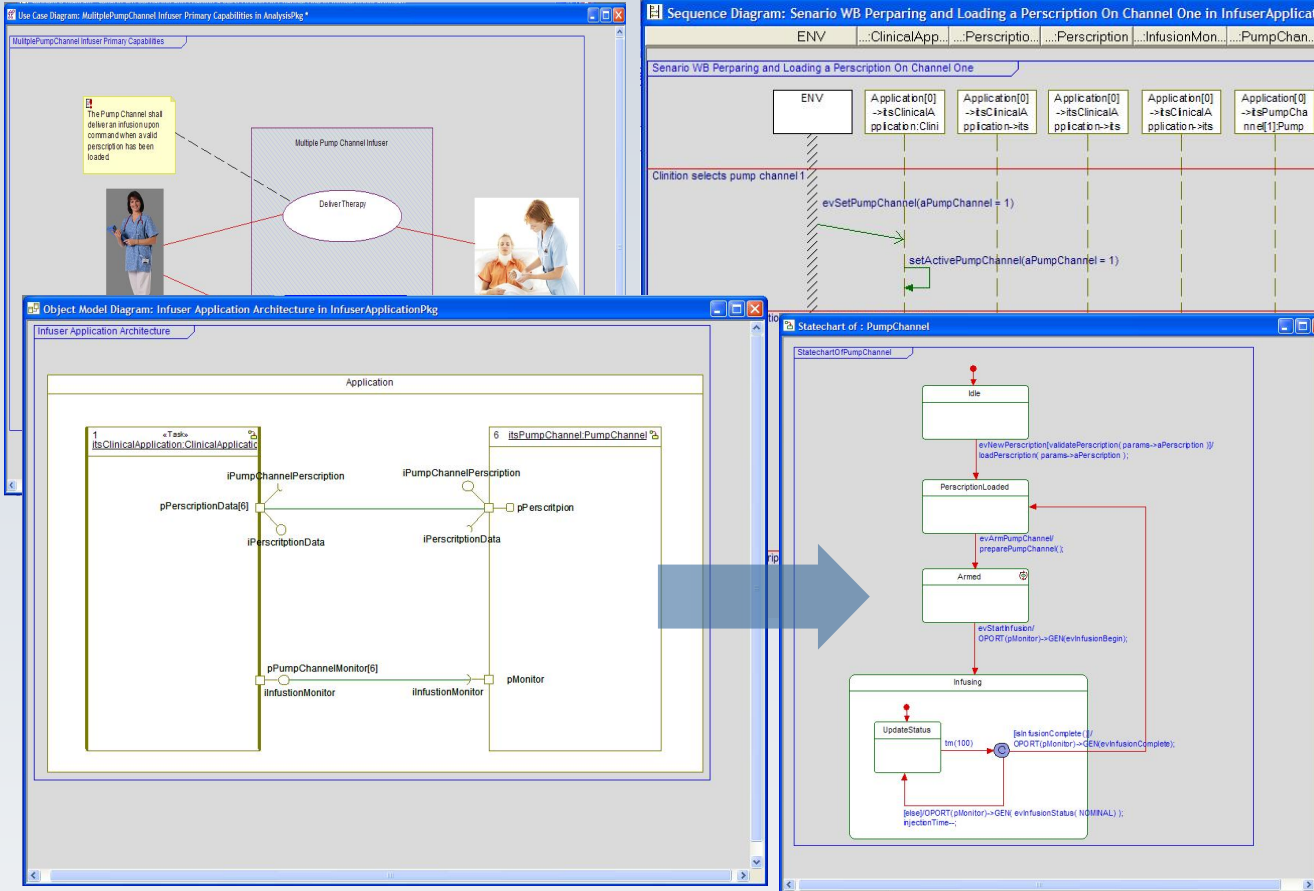


К MDD можно двигаться постепенно!

- Новый код разрабатывается на основе моделей
- Имеющийся код и внешние библиотеки подключаются и используются в модели как внешние
- Для модификации имеющийся код трансформируется в модель и изменяется уже на уровне модели

Разработка нового кода на основе моделей

Визуальное моделирование на UML 2.1



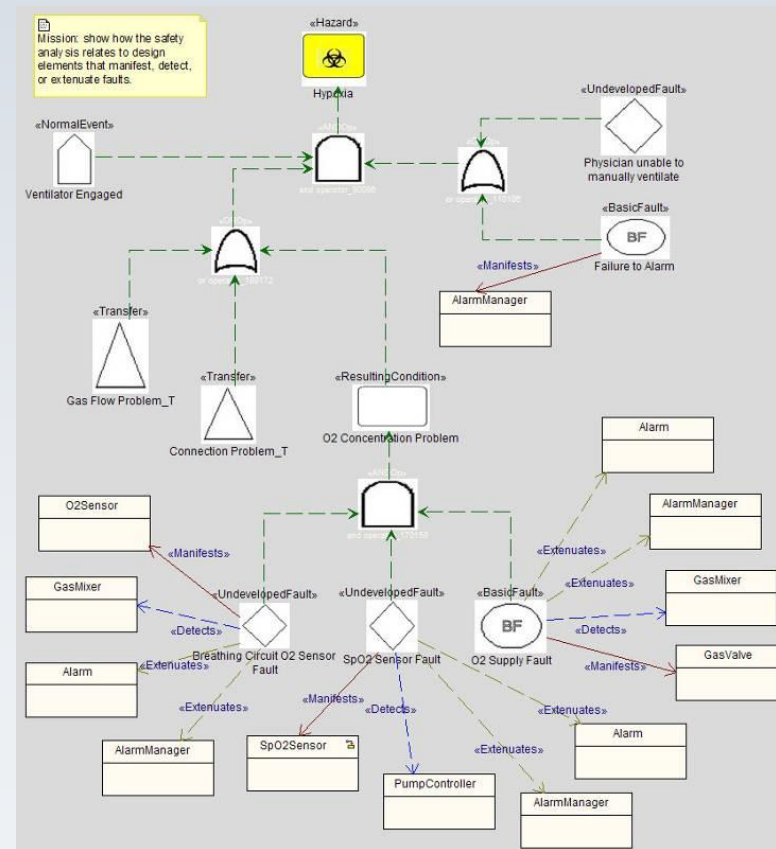
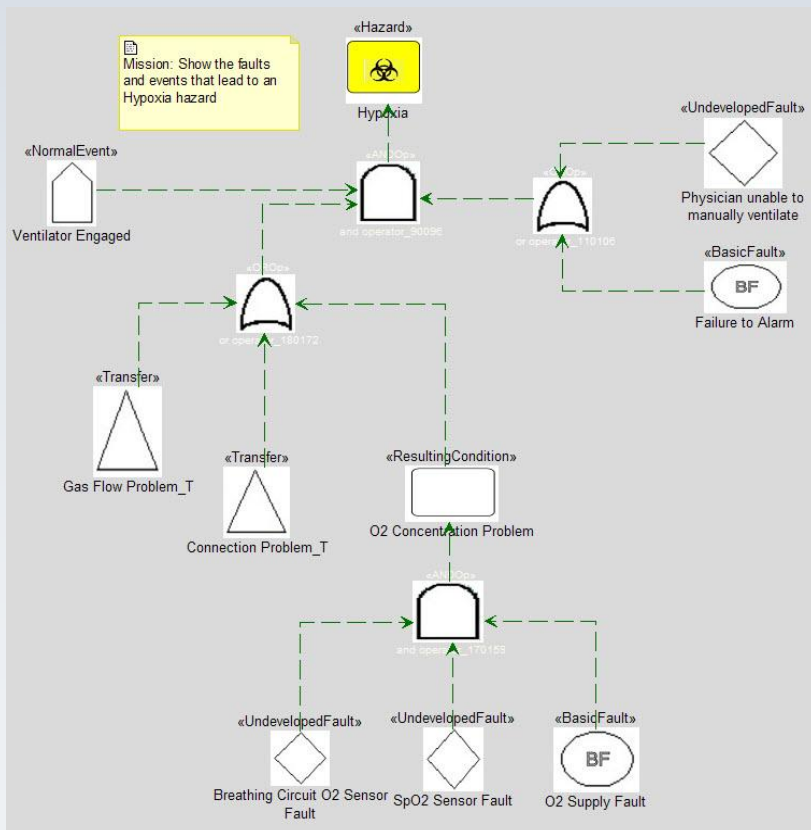
- Визуализация:
 - Требований
 - Структуры
 - Взаимодействия
 - Поведения
 - Ограничений
- Улучшение коммуникаций
- Согласованная информация в различных представлениях
- Enhanced Collaboration
- Стандартный, формальный язык
 - Непротиворечивый

“Rhapsody - лучшее решение для разработки встраиваемых систем, поддерживающее UML 2.1.

Источник: “Reducing OEM Development Costs and Enabling Embedded Design Efficiencies Using the Unified Modeling Language (UML 2.0)”, Embedded Market Forecasters

Создание языков предметной области

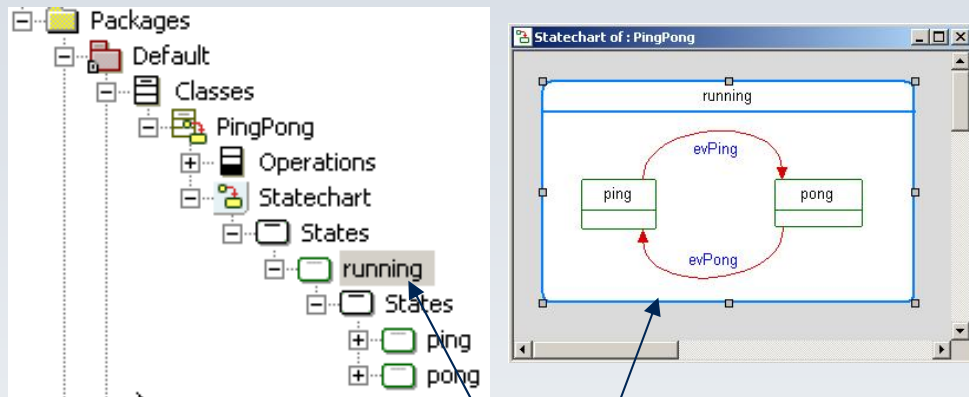
- Адаптация языка и среды разработки для вашей предметной области
- Различные профили позволяют создавать специализированные диаграммы, и эффективно моделировать *на языке предметной области*
- Например, Functional C для моделирования приложений на C или профиль Safety Analysis для анализа надежности и безопасности систем



FTA-
диаграммы из
профиля
Safety Analysis

Семантическая проверка формальных моделей

- Обнаружение ошибок в формальных моделях на основе правил
- Выбор правил для проверки
- Разработка собственных правил



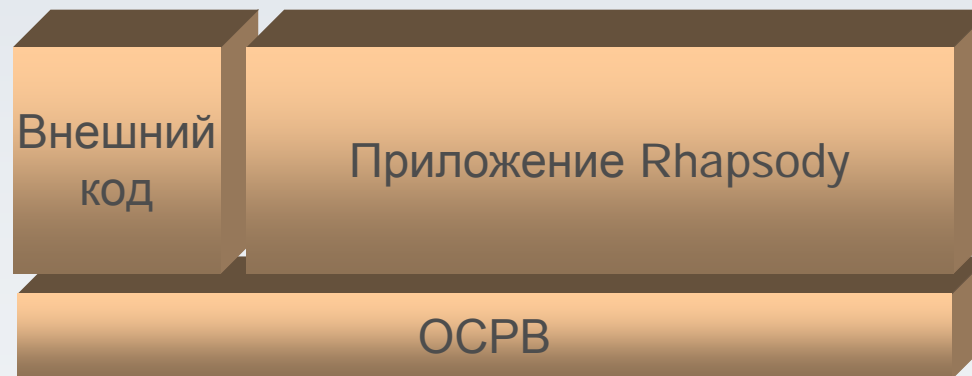
Checks	Domain	Integrity
[-] Errors (1)		
[-] Or state with no default state (1)	Statechart	Complete
[-] Warnings (9)		
[+] Empty description (7)	Common	Complete
[+] Isolated states (2)	Statechart	Complete
[+] Info (1)		
[+] Element with no relations (1)	Class Model	Complete

The image shows a configuration dialog titled "Configuration : PingPongWin32 in DefaultComponent". It has tabs for General, Description, Initialization, Settings, Checks, Relations, Tags, and Properties. The "Checks" tab is active, showing a list of checks to be performed. The list includes various checks with their names, domains, severities, and integrities.

Name	Domain	Severity	Integrity
<input checked="" type="checkbox"/> A <<CORBAInterface>> is mapped to server cod...	Common	Warning	Correct
<input checked="" type="checkbox"/> A COM Server/COM Library can contain at most ...	Common	Error	Correct
<input checked="" type="checkbox"/> A COM TLB component can contain only one pa...	Common	Error	Correct
<input checked="" type="checkbox"/> Activity diagram contains unsupported elements, ...	Common	Warning	Correct
<input checked="" type="checkbox"/> Attribute will not be accessible from the Web bec...	Common	Warning	Complete
<input checked="" type="checkbox"/> Code generation for operations with activity diagr...	Common	Warning	Complete
<input checked="" type="checkbox"/> Component contains CORBA elements, but confi...	Common	Error	Correct
<input checked="" type="checkbox"/> Constructors and destructors cannot be exported ...	Common	Warning	Complete
<input checked="" type="checkbox"/> Default names	Common	Warning	Complete
<input checked="" type="checkbox"/> Dependency between components will not be ge...	Common	Warning	Correct
<input checked="" type="checkbox"/> Empty description	Common	Warning	Complete
<input checked="" type="checkbox"/> Only components stereotyped as COM DLL, CO...	Common	Error	Correct
<input checked="" type="checkbox"/> Operation with activity diagram contains user-sup...	Common	Warning	Correct
<input checked="" type="checkbox"/> Package is defined under its class in the same im...	Common	Error	Correct
<input checked="" type="checkbox"/> The active component is defined with "Other" as ...	Common	Error	Correct
<input checked="" type="checkbox"/> Web support is not available for a language varia...	Common	Warning	Complete

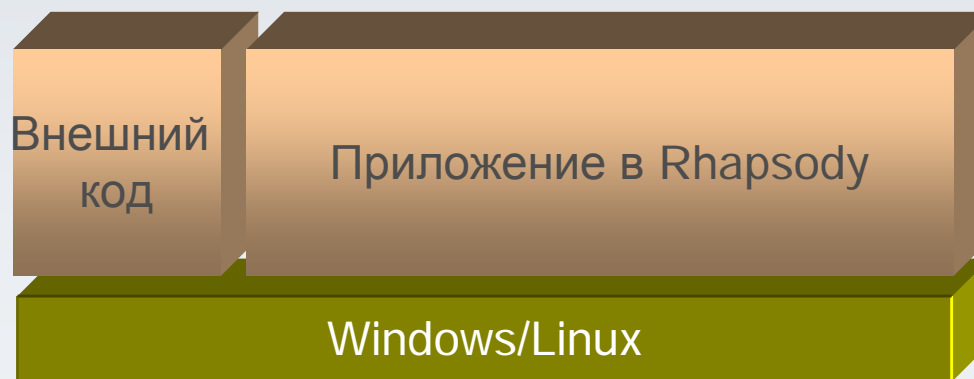
Генерация программного кода на основе моделей

- Генерация программного кода на языках C, C++, Java и Ada
 - Генерация поведенческого кода, а не только структуры
 - Генерация сборочных файлов для различных SDK
 - Генерация проектов для Eclipse CDT и Visual Studio
 - Генерация чистого, понятного кода, удобного для редактирования и отладки
 - Генерация кода в режиме анимации для тестирования на прикладном уровне



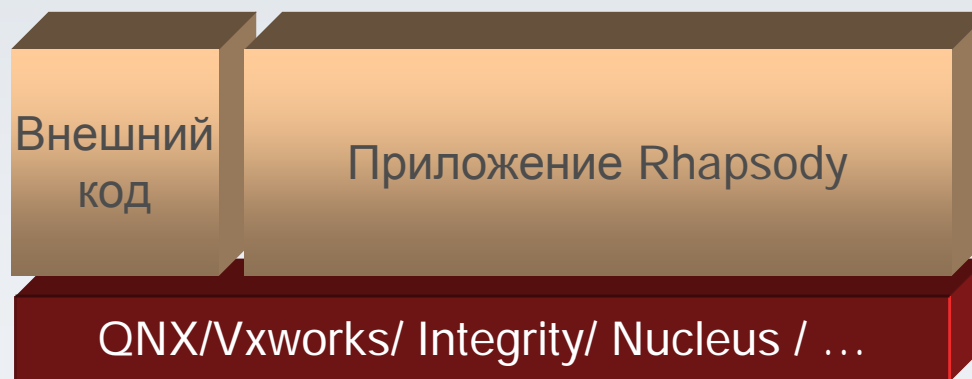
Проверка приложения на машине разработчика

- Разработка ПО до момента готовности аппаратуры
- Создание работающих прототипов для проверки на ранних этапах
- Быстрый перенос приложения на любую целевую платформу



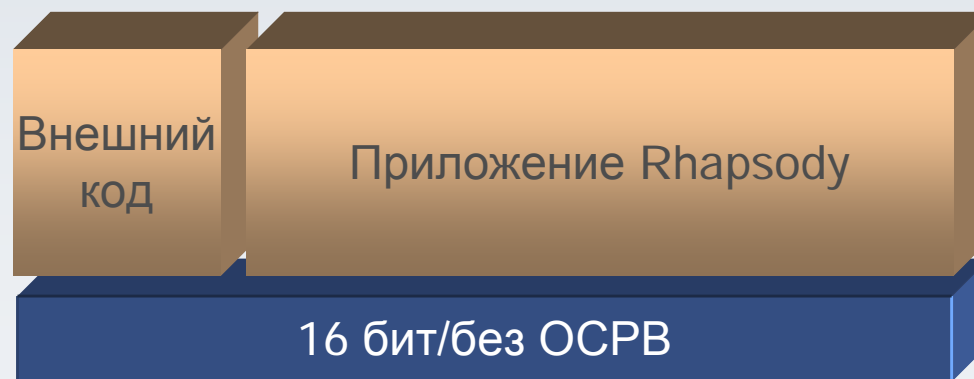
Перенос на целевую платформу

- Быстрый перенос вашего проекта на любую целевую платформу
- Поддерживаются многие ОС
- Кастомизация для поддержки новых ОС



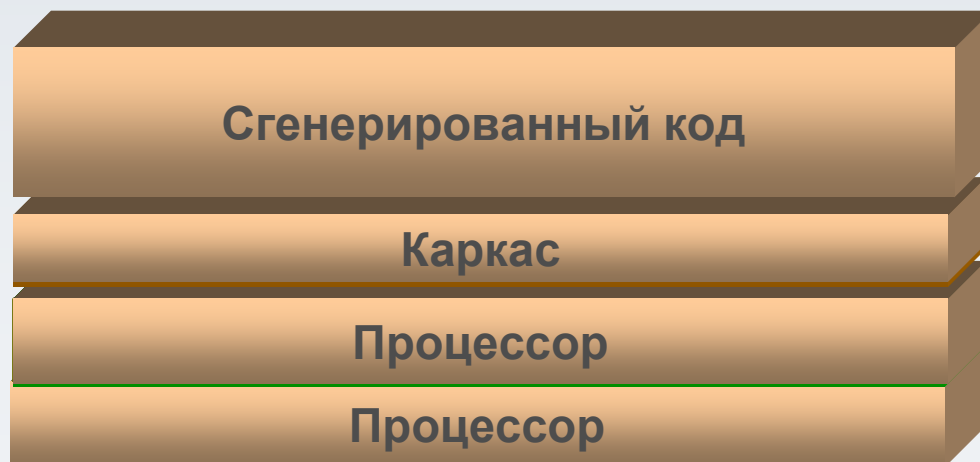
Поддержка устройств с ограниченными ресурсами

- Быстрый перенос кода целевые платформы с ограниченными ресурсами
- Минимизация размера исполняемого файла
- Контроль за управлением памятью и исполнением задач



Генерация кода на основе каркаса приложений реального времени

- Каркас предоставляет базовые классы, реализующие концепции UML, и используемые в сгенерированном коде
- Подключается при сборке приложения в виде статических библиотек
- Абстрагирует сгенерированный код от платформы
- Доступны реализации каркаса в исходном коде для большинства встраиваемых платформ
- Доступны различные урезанные реализации каркаса (OXF, IDF, Synchronous)



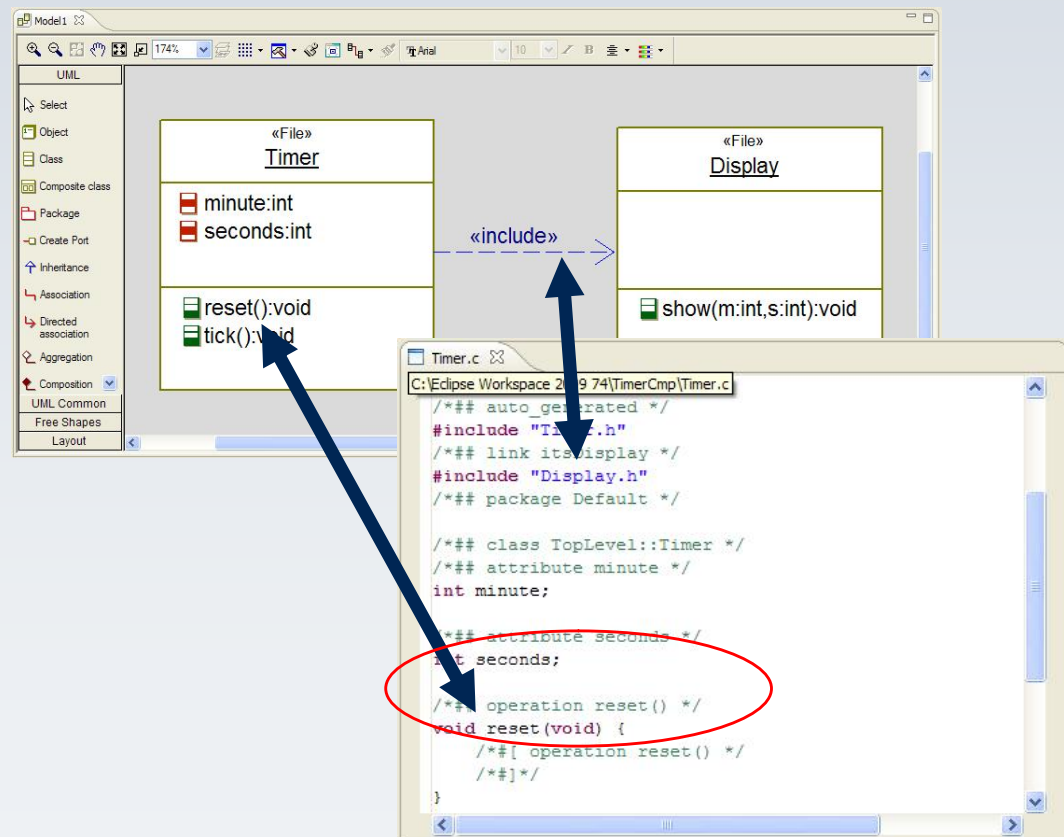
Каркас состоит из двух частей:

Object Execution Framework
(OXF)

Animation and Tracing
Framework

Динамическая синхронизация модели и кода

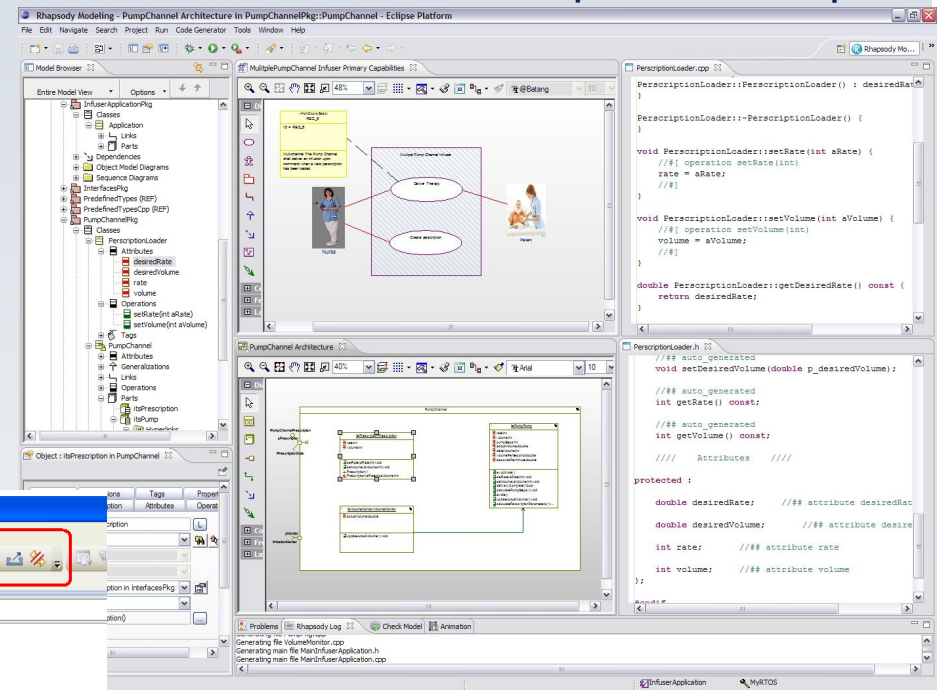
- Работайте на уровне кода или модели
 - Уменьшает время изучения
 - Увеличивает вашу эффективность
- Модель и код всегда синхронизированы
 - Изменение одного представления ведет к изменению другого
 - Важно для разработки встраиваемого ПО реального времени



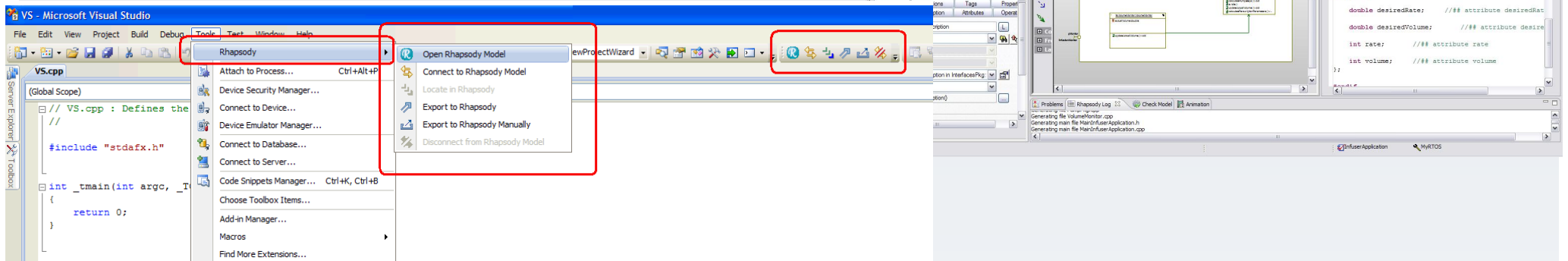
Интеграция с IDE

- Объединение возможностей моделирования с инструментами IDE
- Rational Rhapsody может быть встроен в Eclipse
 - Большинство IDE для встраиваемых ОС основаны на Eclipse
- Внешняя интеграция с Visual Studio
 - Создание и обновление проекта
 - Двухнаправленная навигация между моделью и кодом

Интеграция с Eclipse



Интеграция с Microsoft Visual Studio



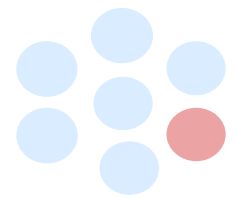
Интеграция Rhapsody в IDE, основанных на Eclipse

- Разработка на основе моделей непосредственно в Eclipse
- Адаптация среды разработки за счет возможностей Eclipse
- Поддержка QNX Momentics, WindRiver Workbench,

Редактор диаграмм Rhapsody

Редактор кода Eclipse

Еclipse - интегрированная среда для проектирования и отладки



- Удобно для разработчиков на основе программного кода
- Выполнение отладки на уровне модели и кода в одной среде

The screenshot displays the Eclipse IDE interface with several key components highlighted by callouts:

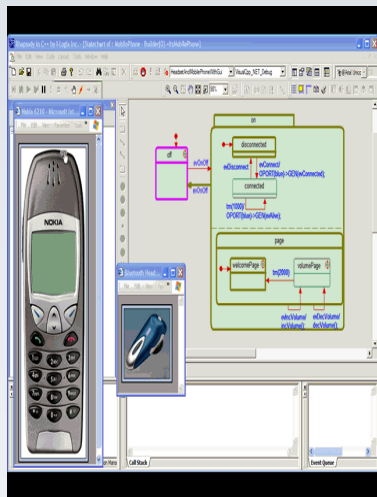
- Отображение модели**: Points to the Package Explorer on the left, showing a project structure with packages like 'Abstract Dishwasher' and 'Acme Dishwasher With Factory'.
- Отображение диаграмм**: Points to the 'Acme Dishwasher Product Line' UML diagram at the bottom, which shows a factory method diagram with 'AbstractFactory' and 'Dishwasher' as interfaces, and 'AcmeFactory' as an implementation. It also shows relationships between 'AcmeTank', 'AcmeJet', and 'AcmeHeater'.
- Использование контекстной подсказки**: Points to a code completion popup for 'AcmeJet::rootState_entDef()' in the main editor, listing methods like 'AcmeFactory', 'AcmeHeater', and 'AcmeJet'.
- Отображение ошибок сборки**: Points to the 'Problems' view on the right, which lists five warnings related to 'Invalid project path: Include path not found'.

Тестирование на уровне модели

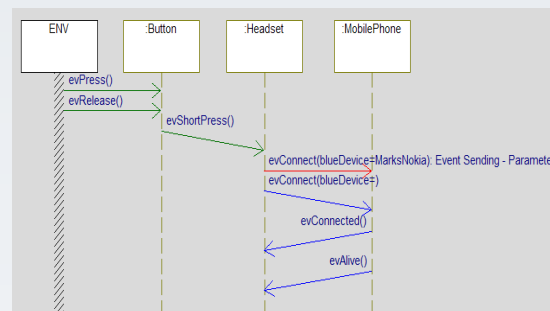
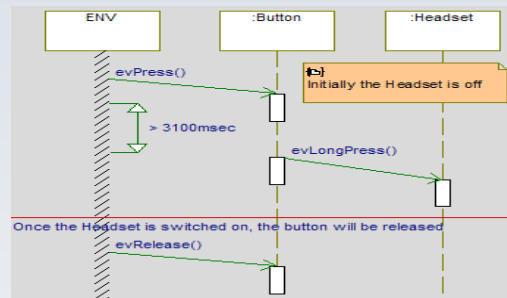
Тестирование на уровне модели

- Привносит преимущества нового уровня абстрагирования и автоматизации для процесса тестирования
- Уменьшает количество ошибок на ранних этапах разработки, когда их исправление не так дорого
- Способствует созданию продуктов удовлетворяющих исходным требованиям

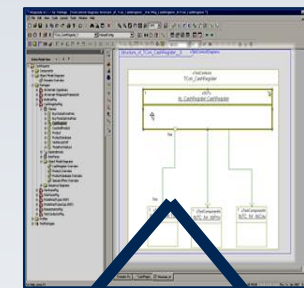
Симуляция



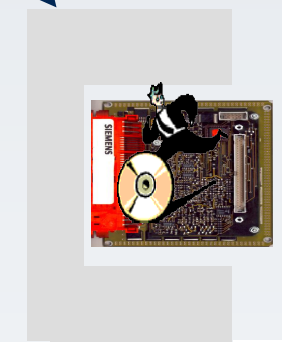
Тестирование на основе требований



Автоматизированное
элементарное
тестирование



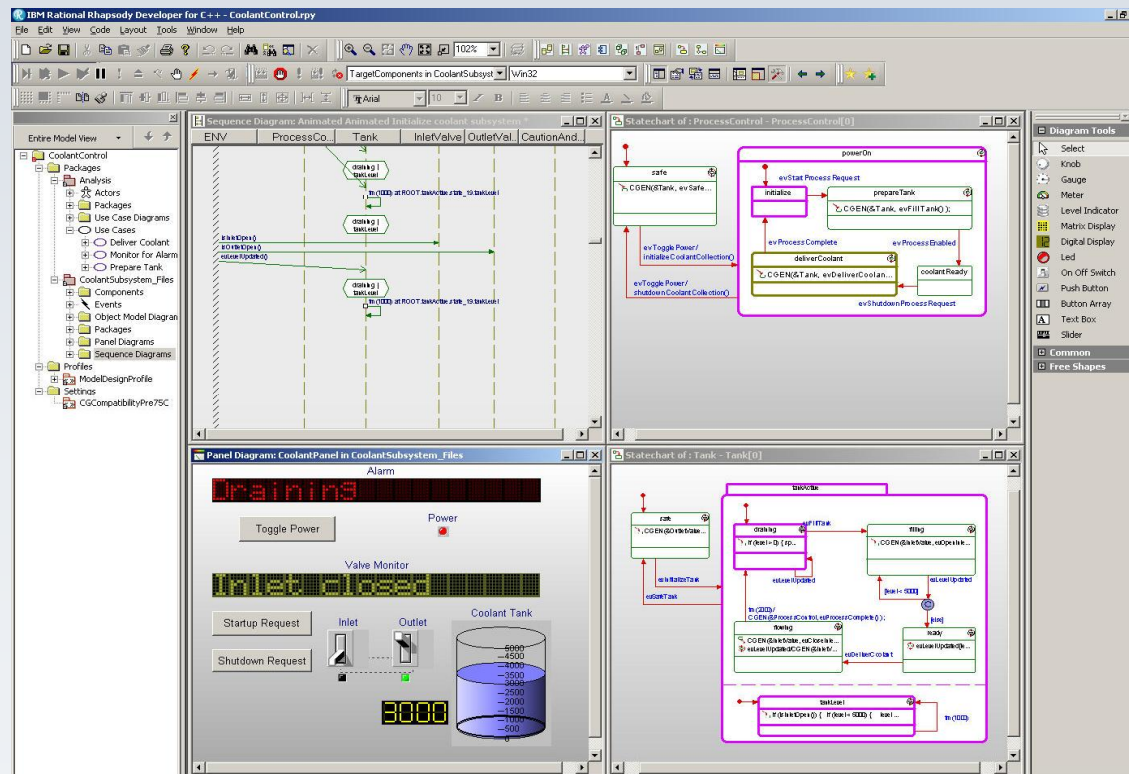
На хосте



На устройстве

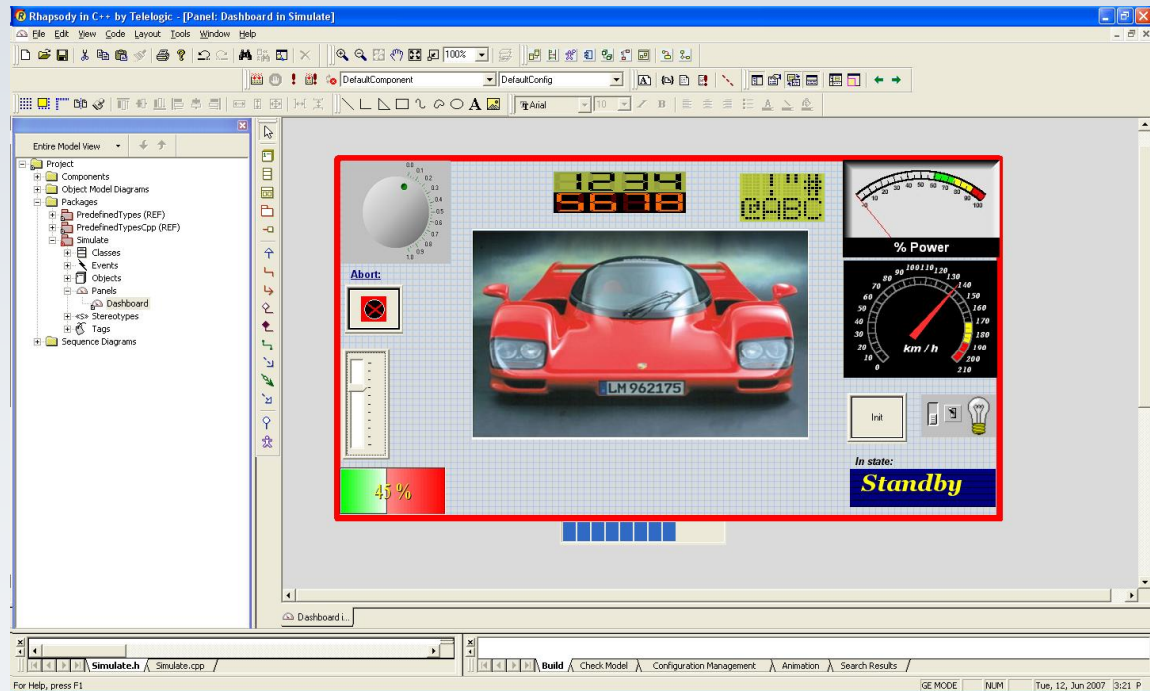
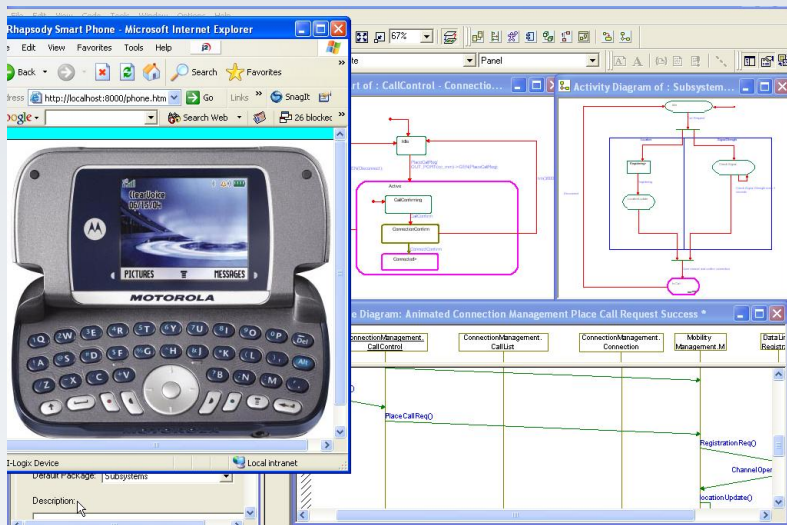
Симуляция, исполнение и анимация

- Исполняйте как можно раньше на хосте для проверки модели
- Исполняйте на целевой платформе с анимацией на хосте для проверки корректности приложения
- Обнаруживайте и исправляйте ошибки раньше, когда их дешевле исправить

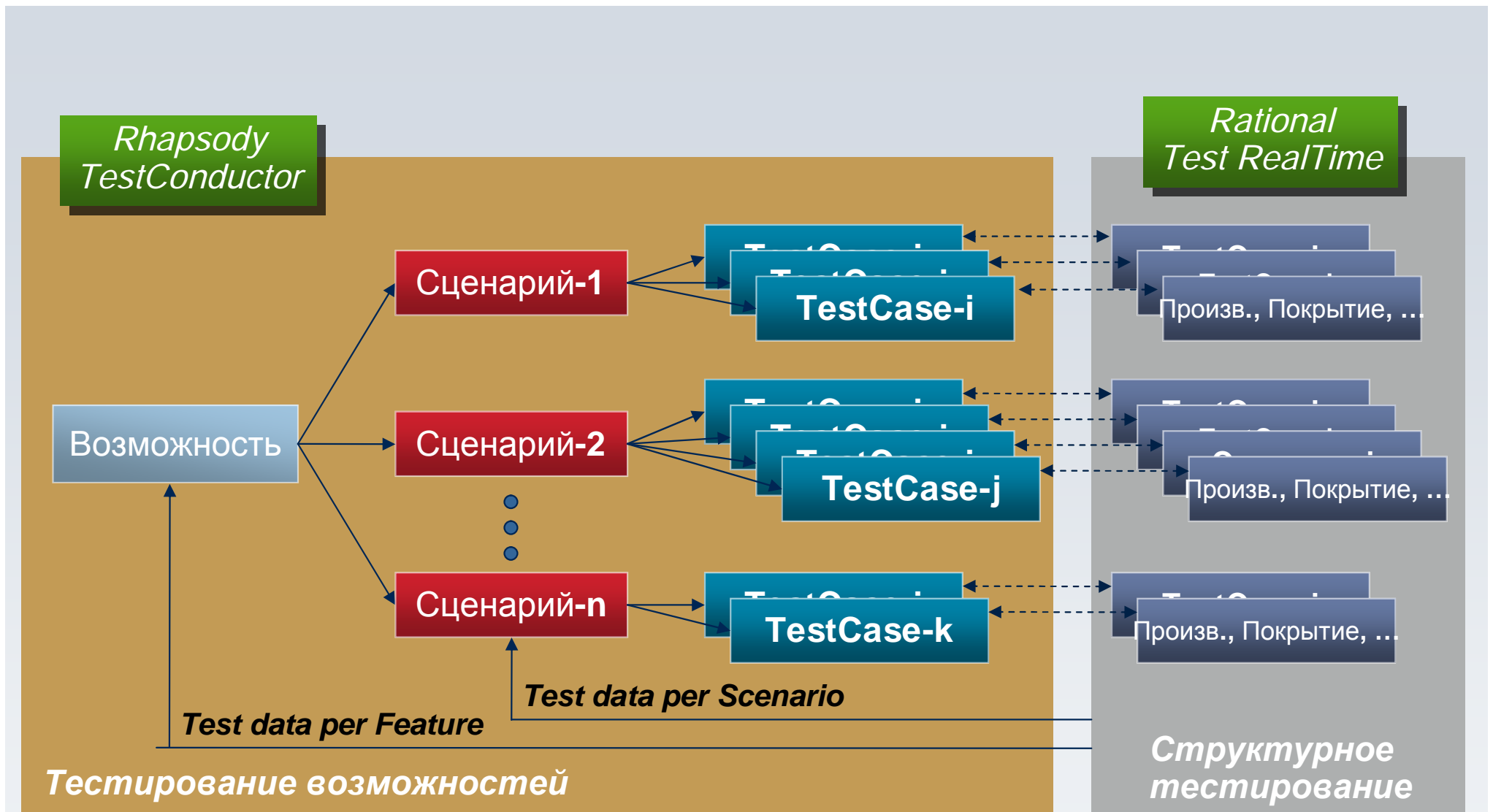


Прототипирование графических интерфейсов

- Создание прототипов интерфейсов для эффективного взаимодействия с заказчиком и внутри команды
- Изменяйте, отслеживайте и анализируйте значения данных в процессе симуляции чтобы убедиться в правильности поведения системы уже на ранних этапах работ



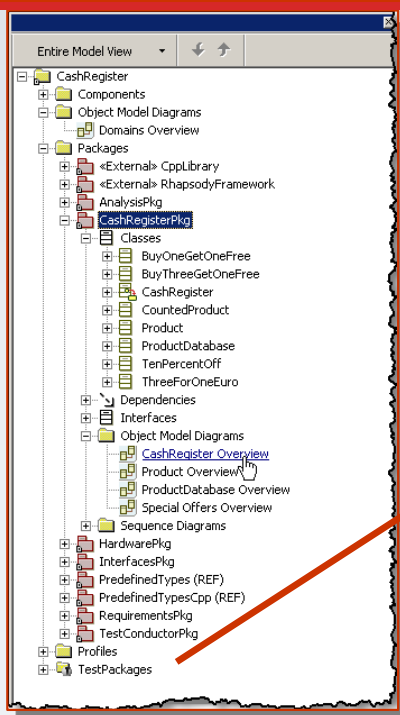
Решение IBM для тестирования



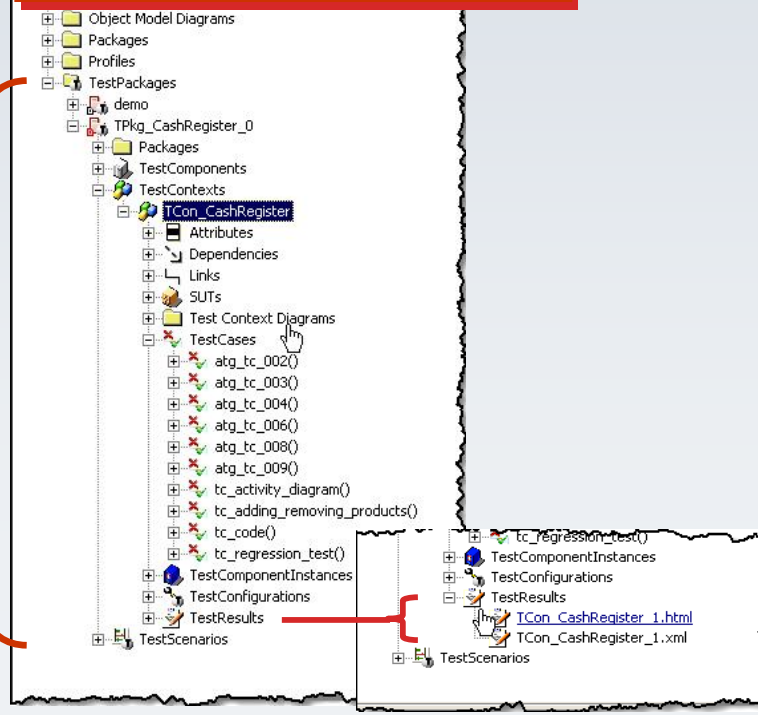
Интеграция процессов проектирования и тестирования

- Интегрированный процесс, основанный на UML2.0 Testing profile
 - Требования связаны с тест-кейсами
 - Лёгкая навигация между артефактами проектирования и тестирования
 - Проект и тесты всегда синхронизированы
 - Автоматически генерируемые отчеты по прогону тестов

Артефакты проектирования



Артефакты тестирования



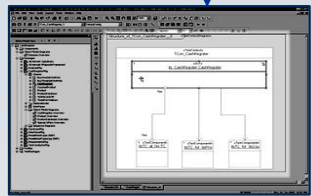
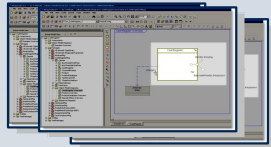
Test Context Result

Отчеты по тестированию

Test Context Result	
Test executed on machine: NBOSC-21-1	
Test executed by user: ubrockmeyer	
Used OS version: Windows 2000 / Windows XP	
Used Rhapsody version: Aries, build 799102	
Used TestConductor version: 2.0, build 530	
Tested Project	
Project:	CashRegister
Active Component:	TCon_CashRegister_5
Active Configuration:	DefaultConfig
Test Context: TCon_CashRegister Summary: PASSED	
tc_code	PASSED
tc_activity_diagram	PASSED
tc_adding_removing_products	PASSED
tc_regression_test	PASSED
atg_tc_008	PASSED
atg_tc_009	PASSED
atg_tc_006	PASSED
atg_tc_002	PASSED
atg_tc_003	PASSED
atg_tc_004	PASSED

Прогон тестов на хосте и целевом устройстве

Результаты проектирования



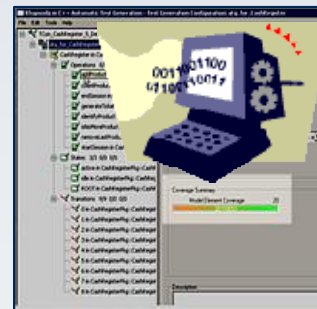
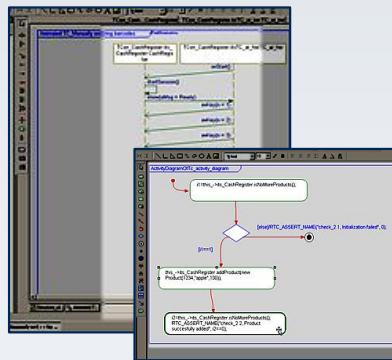
Сгенерированное
тестовое
окружение

**Профиль UML для
тестирования**

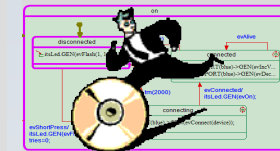
Визуальные
тест-кейсы

Сгенерированные
тест-кейсы

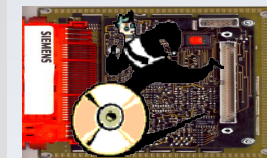
Тест-кейсы в коде



На хосте



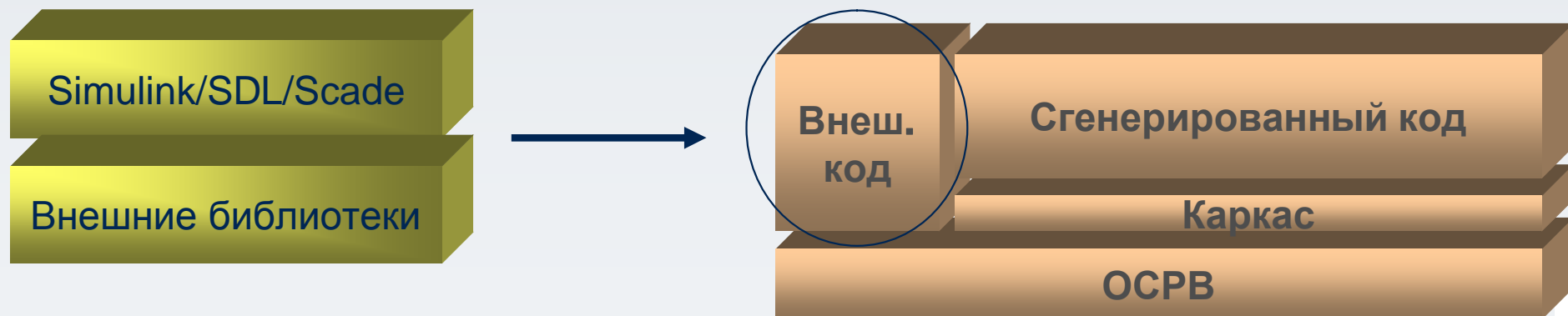
На устройстве



Использование имеющегося кода

Использование имеющегося кода

- Отображение кода в модели как внешнего
 - Появляется возможность визуально использовать внешний код в модели
 - При сборке приложения код не генерируется, а подключается как внешний
 - Последовательный переход к процессу разработки на основе модели
- Полная трансформация кода в модель
 - Предоставляет все преимущества разработки на основе моделей

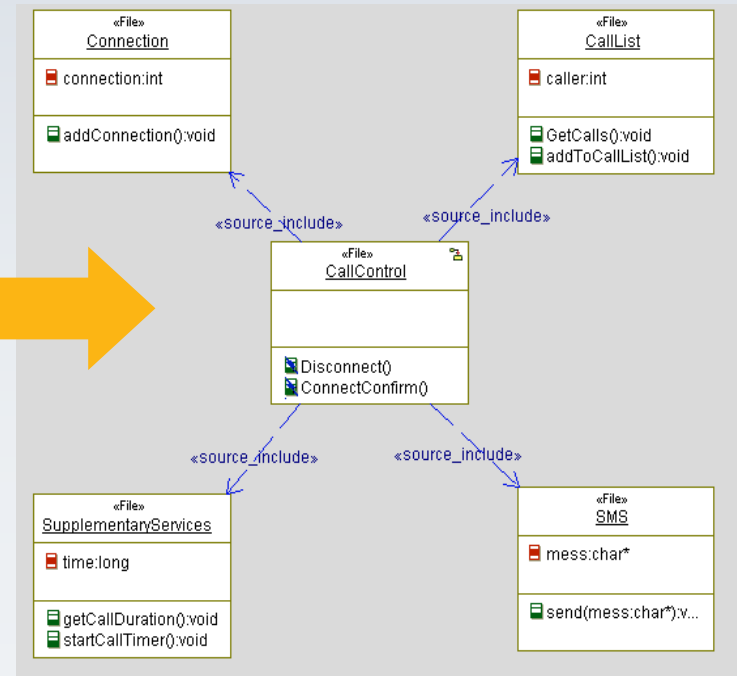


Визуализация кода как внешнего

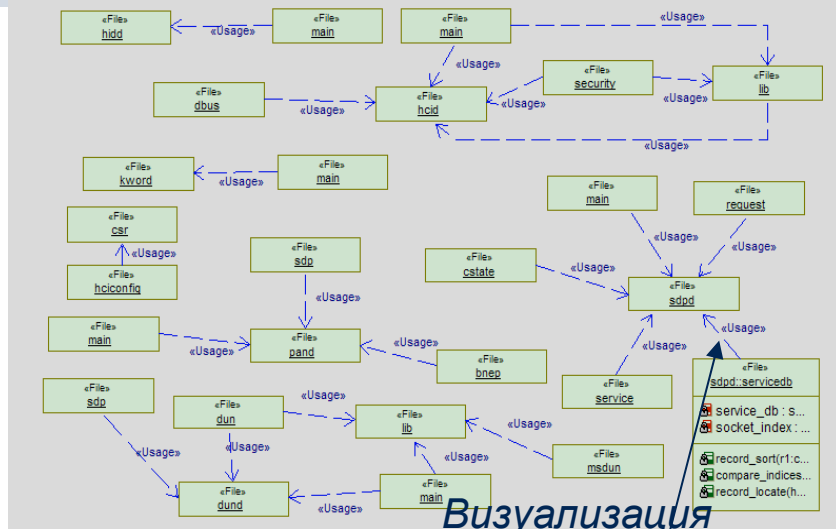
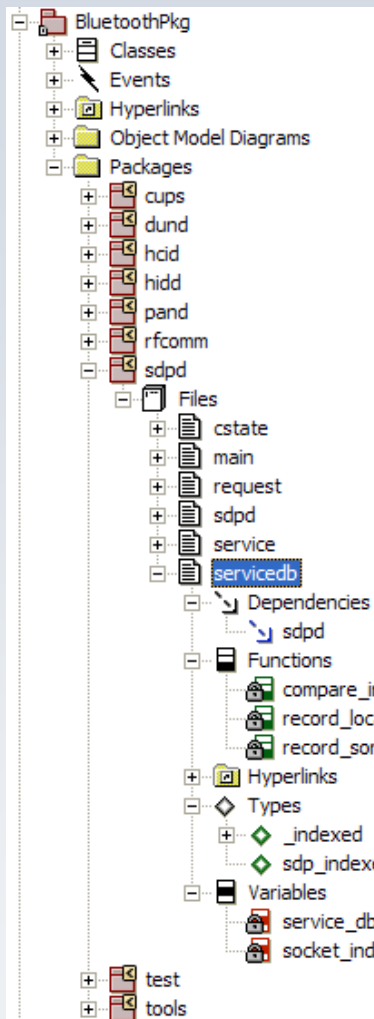
- Повышает понимание имеющихся наработок
- Позволяет автоматическое документирование, уменьшая время до выпуска продукта
- Позволяет работать с людьми, программирующими вручную
- Повторное использование существующих наработок с лучшим пониманием



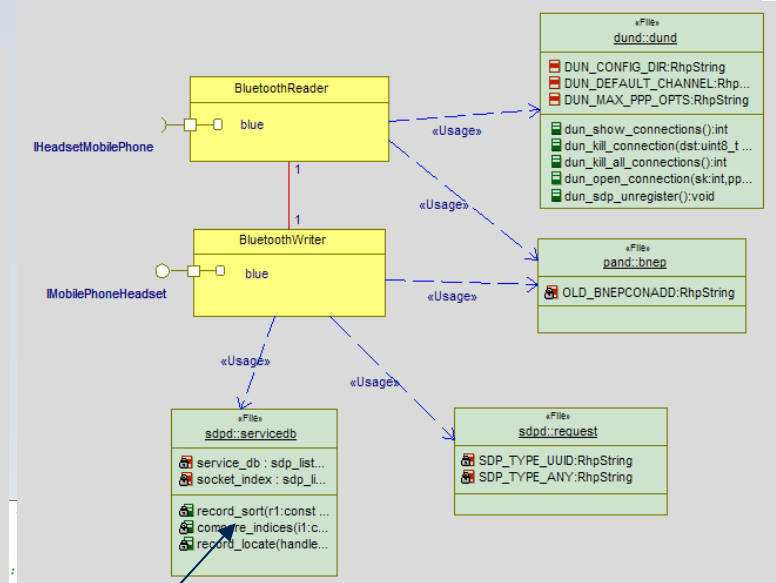
```
.....
BspBody in C : 7.2
  Login      : userpc
  Component  : SystemBuild
  Configuration : Target
  Model Element : CallList
  Generated Date : Wed, 9, Jan 2009
  File Path   : SystemBuild\Target\Architecture\
.....
#include Architecture_ConnectionManagement_CallList.h
#define Architecture_ConnectionManagement_CallList
#include <os/Rtc.h>
#include "Architecture_ConnectionManagement/Connection.h"
.....
/* package Architecture::ConnectionManagement */
/* Maintains a list of currently active calls */
/* class TopLevel::CallList */
/* Counter which keeps track of number of active calls
extern int callList;
.....
BspBody in C : 7.2
  Login      : userpc
  Component  : SystemBuild
  Configuration : Target
  Model Element : Connection
  Generated Date : Wed, 9, Jan 2009
  File Path   : SystemBuild\Target\Architecture\
.....
#include Architecture_ConnectionManagement
#include <os/Rtc.h>
#include "Architecture_ConnectionManagement/Connection.h"
.....
/* package Architecture::ConnectionManagement */
/* Maintains a list of currently active calls */
/* class TopLevel::Connection */
/* attribute connection */
int connection;
void addConnection() {
  /* If state addConnection().ROOT.IncrementConnection.(Entry) */
  connection++;
}
.....
/* Methods Implementation */
void CallControl_Init(RtcTask * p_task) {
  RtcReactive_List((CallControl_rtc_reactive), (void*)CallControl_p_task, CallControl_rtc_reactive);
  RtcReactive_Selector((CallControl_rtc_reactive), RtcFALSB);
  isRMT = NULL;
  CallControl_InitStateSelect();
}
void CallControl_CleanUp() {
  RtcReactive_CleanUp((CallControl_rtc_reactive));
}
int CallControl_rootState_IN(const struct CallControl_s* const me) {
  return 1;
}
static void CallControl_rootState_endOf(void * const void_ptr) {
  CallControl_rootState_subState = CallControl_idle;
  CallControl_rootState_active = CallControl_idle;
}
static RtcReactiveStatus CallControl_rootState_dispatchEvent(void * const void_ptr, short id) {
  RtcReactive_Selector res = eventOnCommand;
  switch (CallControl_rootState_active) {
    case CallControl_idle:
      if(id == PlaceCallReq_ConnectionManagement_Architecture_id) {
        /* Transition 2 */
        OSN((CallControl, PlaceCallReq));
      }
      CallControl_Active_endOf();
      res = eventOnCommand;
      break;
}
}
.....
#endif
.....
File Path : SystemBuild\Target\Arc
```



Пример использования внешнего кода



Визуализация



Подключение

```

#ifndef HAVE_CONFIG_H
#include <config.h>
#endif

#include <malloc.h>
#include <syslog.h>
#include <sys/socket.h>

#include <bluetooth/bluetooth.h>
#include <bluetooth/l2cap.h>
#include <bluetooth/sdp.h>
#include <bluetooth/sdp_lib.h>

#include "sdpd.h"

static sdp_list_t *service_db;

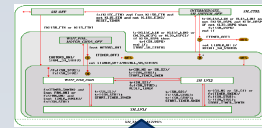
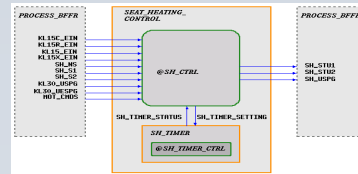
/*
 * Ordering function called when inserting a service record.
 * The service repository is a linked list in sorted order
 * and the service record handle is the sort key
 */
static int record_sort(const void *r1, const void *r2)
{
    const sdp_record_t *rec1 = (const sdp_record_t *)r1;
    const sdp_record_t *rec2 = (const sdp_record_t *)r2;

    if (!rec1 || !rec2) {
        SDPFERR ("NULL RECORD LIST FATAL\n");
        return -1;
    }

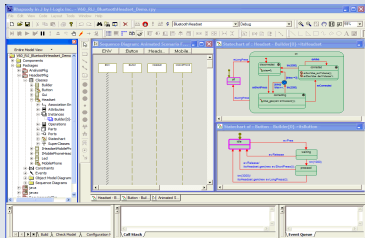
    return rec1->handle - rec2->handle;
}
    
```

Rhapsody как интегрирующий каркас

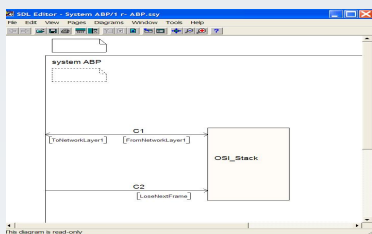
Модель Statestate Функциональный



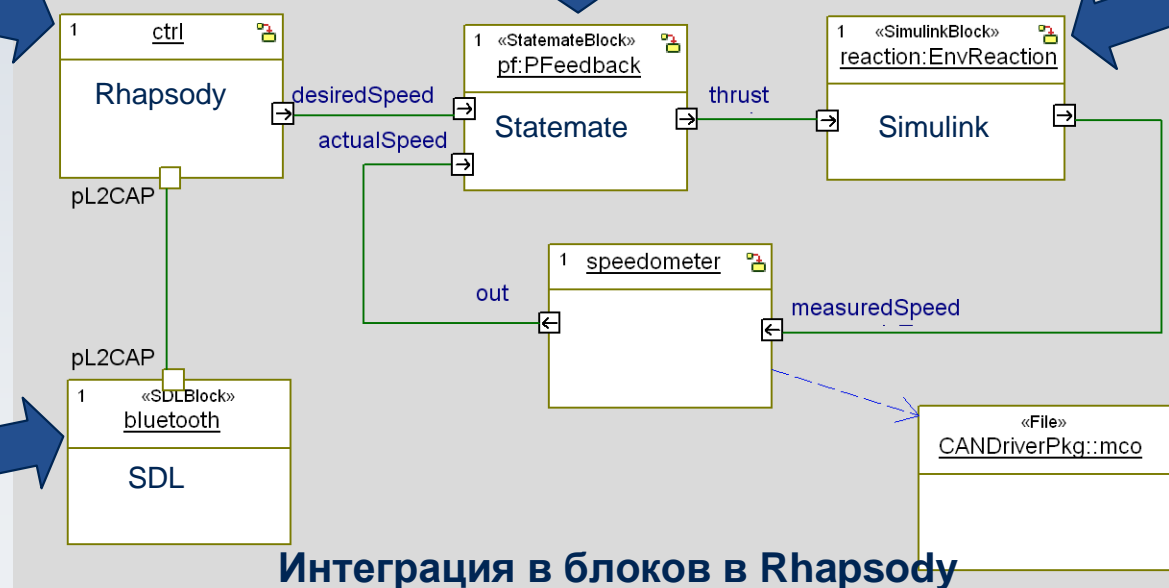
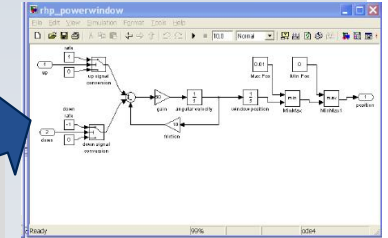
Модель в Rhapsody Логический алгоритм



SDL Suite Model Реализация протокола



Модель Simulink Вычислительный алгоритм



Алгоритмы,
реализованные
во внешнем
коде

- Предоставляет интегрированное решение для сложных задач
- Используйте язык предметных областей и лучшие инструменты для работы

Модификация имеющегося кода

Обратное проектирование имеющегося кода с целью модификации на уровне модели

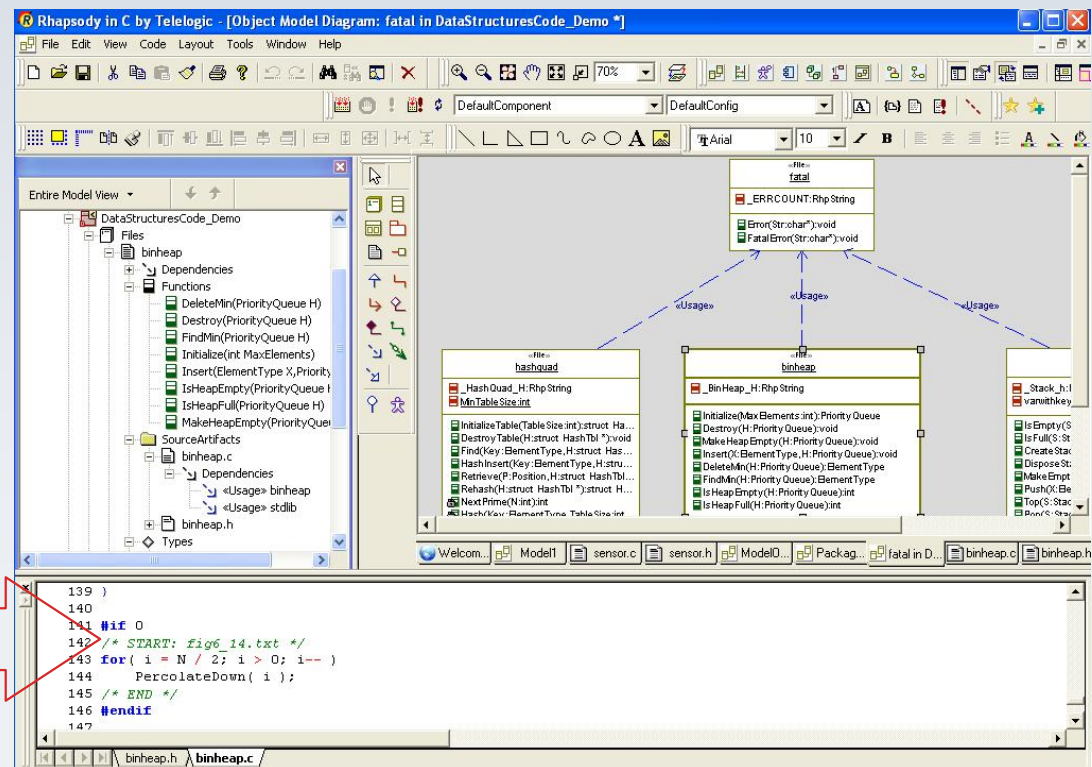
- Постепенный, итеративный процесс
- Не есть самоцель, а средство для изучения и модификации существующего кода
- В результате получается полноценная модель, на основе которой генерируется программный код
- Добавляет возможности отладки и тестирования существующего кода на уровне модели

Уважение к коду при обратном проектировании

- Динамическая синхронизация модели и кода
 - Уважение к коду при обратном проектировании позволяет сохранять структуру, имена, положение и порядок существующего кода
 - Сгенерированный код выглядит очень похожим на исходный
 - Дальнейшие изменения в коде автоматически попадают в модель

```
return H->Size == 0;
}
int IsHeapFull( PriorityQueue H )
{
    return H->Size == H->Capacity;
}
void Destroy( PriorityQueue H )
{
    free( H->Elements );
    free( H );
}
#if 0
/* START: fig6_14.txt */
for( i = N / 2; i > 0; i-- )
    PercolateDown( i );
/* END */
#endif
```

Уважение



Анализ взаимодействия элементов кода

- Модель позволяет эффективно анализировать взаимодействие между элементами кода в различных вариантах использования
- В процессе исполнения кода сохраняются анимированные диаграммы последовательности
- Позволяет получить описание протоколов взаимодействия
- Определяет необходимые интерфейсы между элементами кода
- Является основой для регрессионного тестирования модифицированной модели

Описание поведения элементов кода на основе конечных автоматов

- Визуализация поведения элемента кода путем определения конечных автоматов
- Конечные автоматы налагают ограничения на последовательность вызова операций
- Увеличивают надежность кода и облегчают его тестирование

Гибкое реагирование на изменения

Сбор, визуализация и трассировка требований

- Улучшение понимания сложных требований путем их детализации
- Улучшенные возможности коммуникации для оценки состояния проекта
- Трассировка требований к модели и коду

The screenshot displays the IBM Rational DOORS and Rhapsody Developer for C++ environment. On the left, the DOORS interface shows a list of requirements, with REQ_3 selected. The requirement text is: "The Clinical Application shall select one of six pump channels to program." Below this, the text "IBM® Rational® DOORS" is visible. In the center, the Rhapsody Developer interface shows a structure diagram of the "Clinical Application Architecture" with a requirement element (REQ_3) highlighted in yellow. A blue callout bubble points to this element with the text "Трассировка между требованиями и моделью". Below the structure diagram, the "Active Code View" shows C++ code with a blue callout bubble pointing to a line of code: "InfusionMonitor itsInfusionMonitor[6];" with the text "Требования в сгенерированном коде".

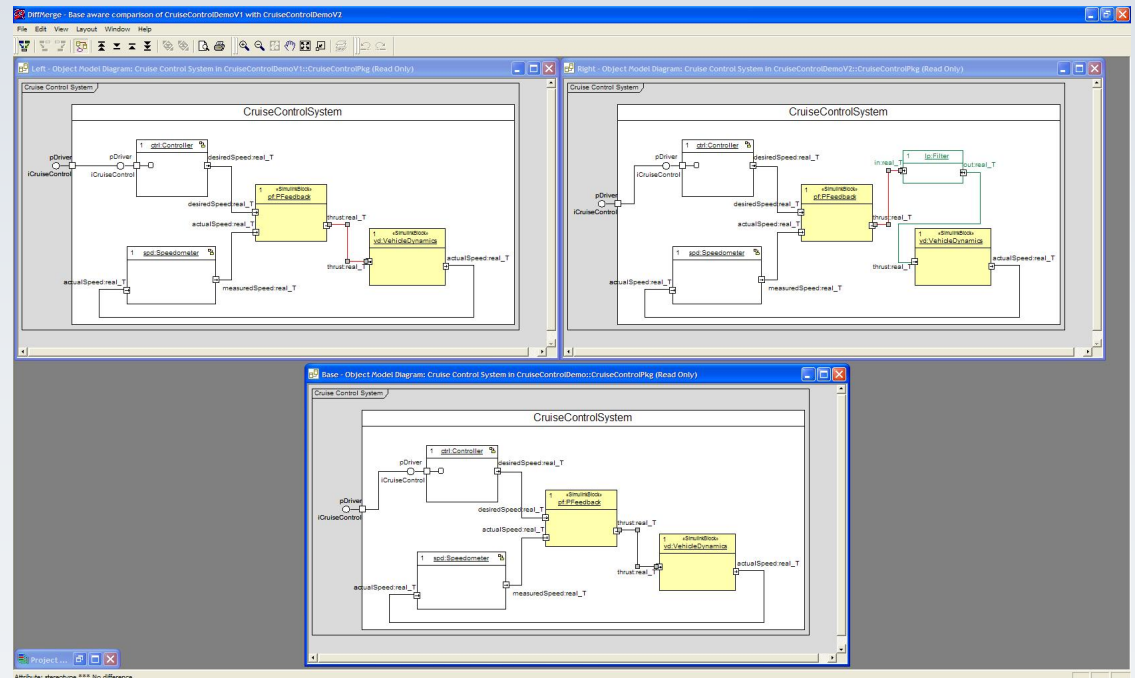
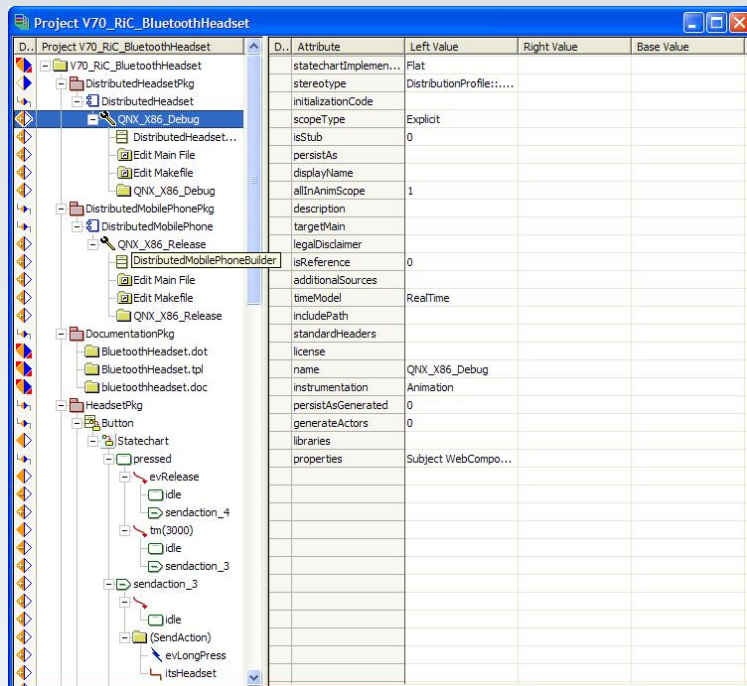
Трассировка между требованиями и моделью

Требования в сгенерированном коде

IBM® Rational® DOORS

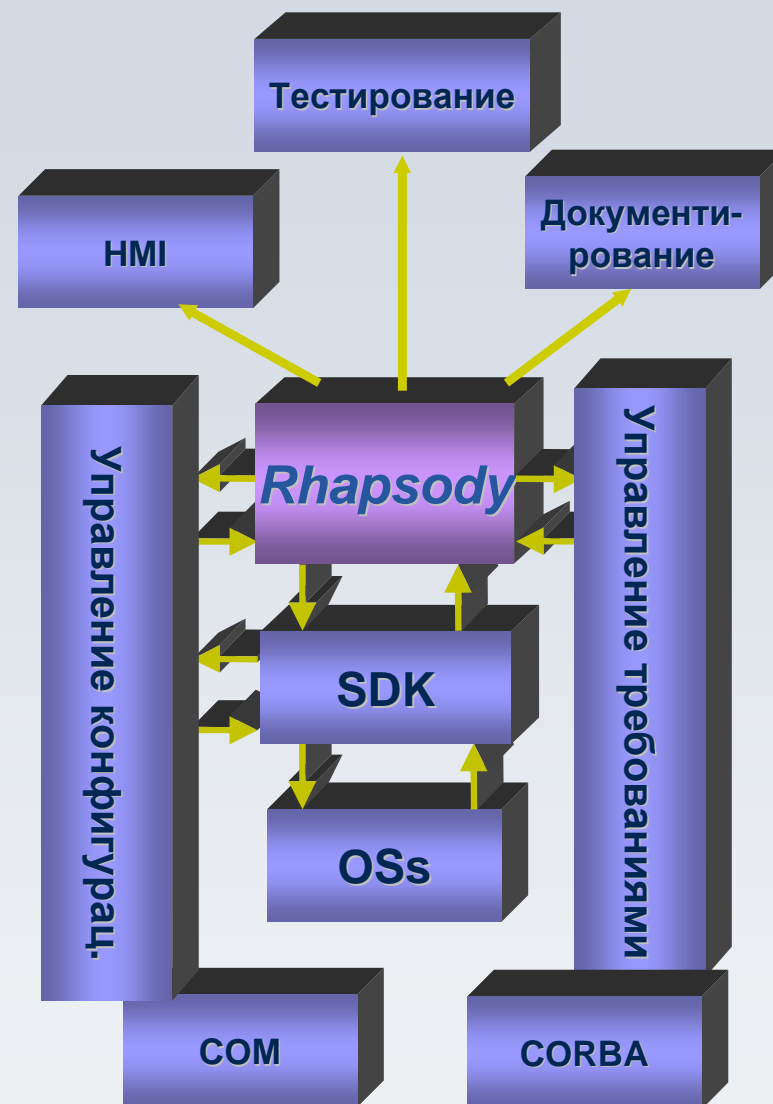
Совместная работа

- Интеграция с системами управления конфигурациями
 - Synergy, ClearCase, и любыми SCC совместимыми системами
- Визуальный анализ изменений и оценка различные проектных решений
- Ручное и автоматическое слияние изменений между несколькими версиями



Открытое решение

- Управление требованиями
- Управление конфигурациями
- Документирование
- Панельная графика / HMI
- Тестирование
- Моделирование непрерывных процессов
- Интегрированные среды разработки (IDE)
- Операционные системы реального времени и стандартные ОС



Поддержка CORBA (Tao) и COM

Возможности IBM Rational Rhapsody для разработки на основе моделей



Возможности IBM Rational Rhapsody

- Решение для моделирования на основе стандартов языков UML/SysML, расширяемое для моделирования предметной области
 - Стандартные формальные модели улучшают взаимодействие
- Гибкие интегрированные возможности для сбора, анализа и трассировки требований
 - Позволяют гарантировать, что система в каждый момент времени удовлетворяет требованиям
- Возможности по тестированию на уровне модели включают симуляцию системы, тестирование на основе требований и автоматическую генерацию тестов
 - Позволяют устранять ошибки на ранних этапах разработки и проверять систему на предмет удовлетворения требованиям

Возможности IBM Rational Rhapsody

- Мощные возможности по генерации кода позволяют полностью создавать приложения на C, C++, Java и Ada
 - Вписываясь в жесткие временные ограничения, создавая приложения быстрее
 - Повторно используя существующий код и модели
 - Легко и быстро создавая приложения для целевой платформы
- Коллективная совместная работа для малых, больших и распределенных команд
 - Позволяя эффективно взаимодействовать и работать в различных командах
 - Увеличивая возможности для повторного использования существующих наработок

Спасибо за внимание!

Дмитрий Рыжов

e-mail: d.ryzhov@swd.ru

<http://www.swd.ru/>



196135, г. Санкт-Петербург,
пр. Юрия Гагарина 23
тел.: (812) 309-2936

115553, г. Москва,
пр. Андропова 22/30
тел.: (495) 651-6136