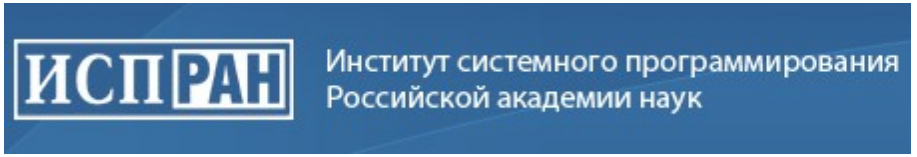
Научно-практический семинар
«Технологии разработки и анализа программ»

# Использование статического анализа для поиска дефектов программного кода

Александр Волков { volkov@ispras.ru }
Сергей Марков { markov@ispras.ru }





ВМК МГУ
17 декабря 2009

# План

- Что такое дефект программного обеспечения
- Методы статического поиска дефектов
- Применение инструментов статического анализа

# Неформальный план

- Что искать?
- Зачем искать?
- Как искать?
- Что со всем этим делать?

# Что такое дефект ПО? Пример

```
 1 void xfoo(xbuf_t *buf, int aid)
 2 {
 3     data_t *ptr = NULL;
 4     if (buf->data) {
 5         ptr = buf->data;
 6     }
 7     stub();
 8     if (aid != -1) {
 9         ptr->id = aid;
10     }
11 }
```

Возможное разыменование нулевого указателя `ptr` при `buf->data !=0` и `aid != -1`

- зависимость от контекста?
- необходимость явного описания предусловия?

# Что такое дефект ПО

- То, что потенциально приводит к проблемам с ПО.
- То, что отрицательно сказывается на качестве ПО.

# Классы дефектов

- Функциональность (поведение)
  - Отказ работы
  - Уязвимости безопасности
  - Масштабируемость
- Сопровождение
  - Архитектура
  - Правила кодирования

# Актуальные виды дефектов функционирования

- разыменование нулевого указателя
- чтение неинициализированной памяти
- выход за границы массива / переполнение буфера
- утечка памяти / ресурсов
- использование освобожденной памяти / ресурсов
- использование невалидированных потенциально вредоносных внешних данных (tainted data / injections)

# Дефекты функционирования (архитектурный уровень)

- Дефекты поведения на уровне логики классов
- Дефекты поведения на уровне взаимодействия параллельных компонент
  - взаимная блокировка
  - состояние гонки

# Актуальность задачи поиска дефектов

- Высокая стоимость ошибки в конечном программном продукте
- Высокая стоимость времени разработчиков для поиска ошибки
- Контроль качества ПО как элемент процесса разработки

# Правила и стандарты

- NIST SAMATE (Software Assurance Metrics And Tool Evaluation)
  - Статический анализ исходного кода программы
  - Сканеры уязвимости веб приложений
  - Динамический анализ исполняемо кода
- MISRA C
  - 1998. "Guidelines for the use of the C language in vehicle based software" — 127 правил
  - 2004 "Guidelines for the use of the C language in critical systems" — 141 правило
- CERT Coordination Center
  - The CERT C Secure Coding Standard
  - The CERT C++ Secure Coding Standard
  - The CERT Sun Microsystems Secure Coding Standarts for Java

# Поиск дефектов: статический и динамический анализ

- Обнаружение несовпадающих множеств дефектов
- Взаимодополняющий контроль качества ПО

# Программный код, используемый для анализа

- Исходный код (язык программирования)
  (Собственный опыт: Си, C++, C#, Java)
  - скриптовые языки
- Исполняемый код
  (Собственный опыт: Java байт-код, Ассемблер для ix86)
  - задача декомпиляции

# Назначение инструментов поиска дефектов

- Анализ исходного кода
  - контроль качества кода системы
  - проверка внесенного/измененного кода
- Анализ исполняемого кода:
  - более точный анализ поведения (?)
  - аудит библиотек / модулей сторонних разработчиков

# Методы статического поиска дефектов

- По абстрактному дереву синтаксиса (AST)
- По графу потока управления (CFG)
- По архитектурным моделям
  - поиск антипаттернов проектирования
- По метрикам
  - вычисление численных характеристик

# Поиск дефектов по AST

- Поиск шаблона AST
- Быстрый анализ
- Довольно легко параметризуется
  - может использоваться для пользовательского поиска синтаксически «интересных» ситуаций

# Поиск дефектов по графу потока управления

- Анализ потоков данных:
  поиск «интересного» фрагмента потока данных
- «Известные» методы:
  - абстрактная интерпретация
  - символические вычисления
- Несколько другие требования, чем к анализу, производимому компиляторами
- Рост производительности вычислительной техники

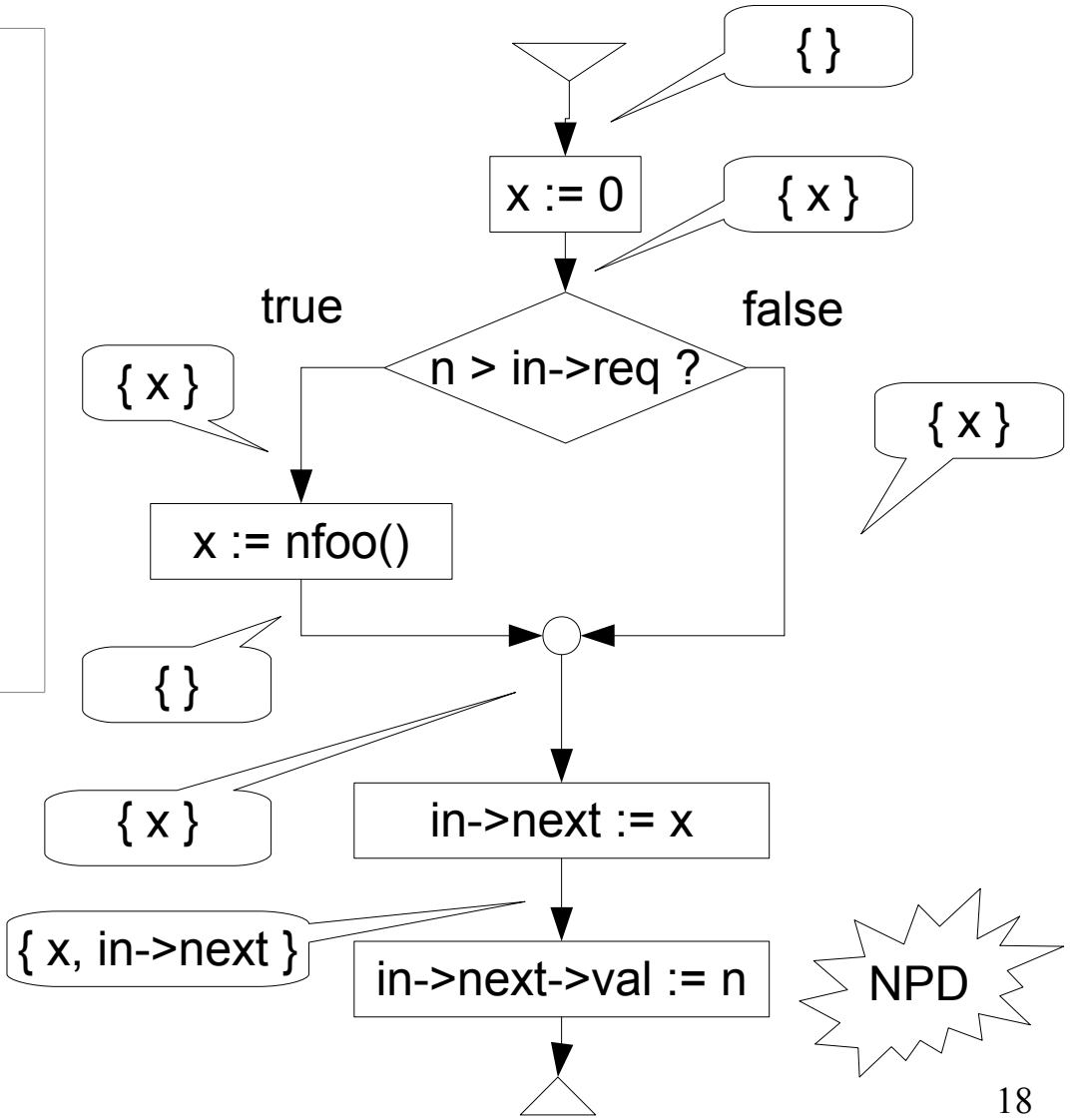
# Поиск дефектов по графу потока управления: Этапы анализа

- построение абстрактного дерева синтаксиса
- сбор семантической информации
- постронение внутреннего представления (Medium-level Internal Representation)
- построение графа вызовов
- запуск анализаторов (checker-ов)

# Распространение абстрактных свойств по графу потока управления

## Пример поиска дефекта

```
 1 void
 2 nsample( xobj_t *in,
 3           int n )
 4 {
 5     xobj_t *x = NULL;
 6     if (n > in->req) {
 7         x = nfoo();
 8     }
 9     in->next = x;
10     in->next->val = n;
11 }
```



# Поиск дефектов по графу потока управления: Необходимые элементы анализа

- Проверка совместности условий пути
  - объединения целочисленных интервалов
  - проверка выполнимости системы формул (SAT-solver)
- Анализ синонимов (alias-анализ)
- Межпроцедурный анализ
- Построение объяснений к дефектам

# Поиск дефектов по графу потока управления:
# Пример: Несовместные условия

```
 1 void u_sample(int t)
 2 {
 3     int v;
 4     int flag;
 5     if (t != -1) {
 6         v = mget(t);
 7     }
 8     stub(t + 1);
 9     if (t != -1) {
10         mfin(v);
11     }
12 }
```

Проверка условия `t != -1` гарантирует, что чтение переменной `v` происходит, только когда она инициализирована

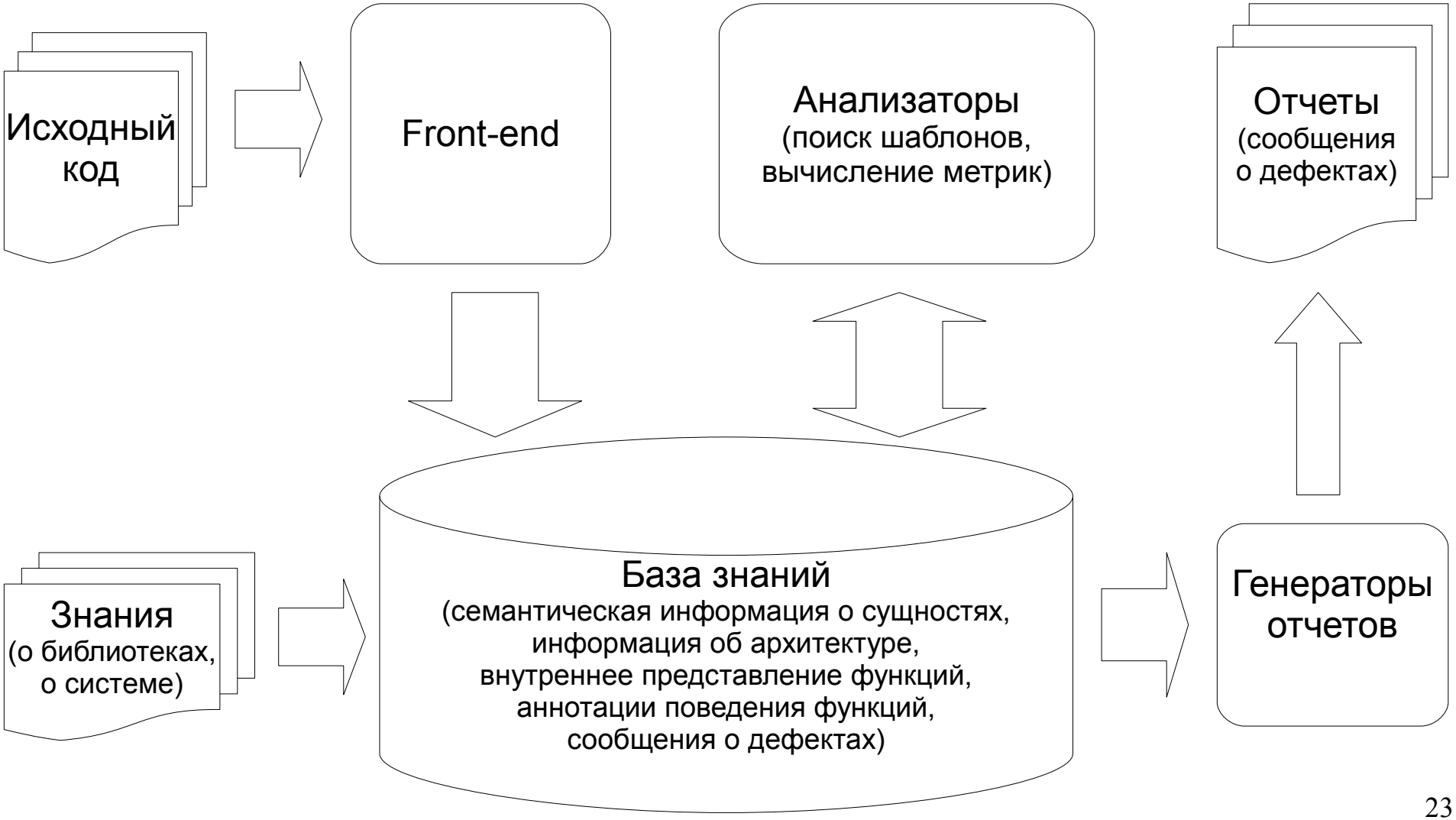
# Поиск дефектов по графу потока управления: Межпроцедурный анализ

- встраивание вызовов функций
  - объем структур данных для анализа
  - рекурсия
- явные аннотации к функциям — «контрактный подход»
  - унаследованный код
  - ошибки в аннотациях
  - поддержание консистентности кода и аннотаций
- автоматическая генерация аннотаций — «метод резюме»
  - точность сгенерированных аннотаций

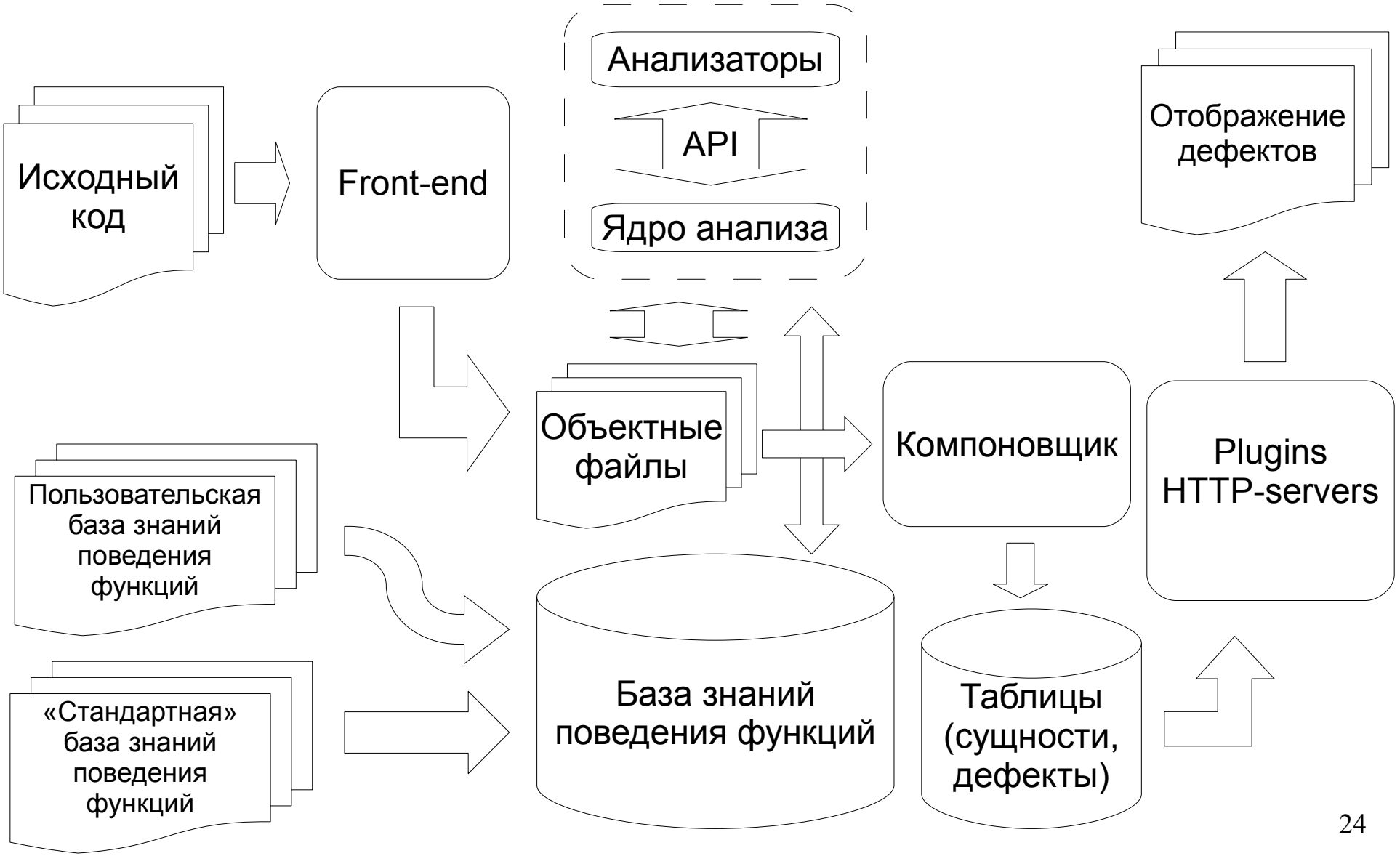
# Поиск дефектов по графу потока управления: Элементы анализа (продолжение)

- Анализ элементов массивов
- Поддержка сложных объектов
- Анализ виртуальных вызовов
- Поддержка исключений
- Контекстно-зависимый анализ

# Архитектура системы статического анализа



# Архитектура Klocwork Insight



# Статический поиск дефектов как экспертная система

- сходные элементы:
  - «база знаний»
  - «подсистема объяснений»
- методы:
  - формальный вывод
  - эвристики
- ошибки:
  - ложные срабатывания (false positives)
  - пропуски (false negatives)

# Параметризация анализа
# Модели описания дефектов

- Императивный подход
  - API для обхода AST
  - Автоматные модели
  - KWPathAPI
- Декларативный подход
  - KAST (аналог XPath)
  - DATALOG (запрос к базе знаний из предикатов)
  - KWPathAPI (использование базы знаний)

# Традиционные дилеммы:

- чувствительность / шум
  (меньше ложных срабатываний /
  меньше пропусков)
- точность / производительность
  (больше достоверных дефектов /
  меньше потребляемых ресурсов)

# Традиционные дилеммы: Приоритеты при решении

- Скорость анализа:
  - за некоторое «обозримое» время
  - за время, сопоставимое с временем компиляции
- Объем кода:
  - программная система
  - драйвер

# Инструменты статического поиска дефектов: Интеграция в пользовательский процесс разработки

- Интеграция в систему сборки проекта
  - задача воспроизведения пользовательской сборки
- Интеграция в систему контроля версий
  - задача отождествления дефектов в различных версиях
- Интеграция в систему отслеживания ошибок

# Некоторые инструменты статического анализа

- lint
- PC-Lint
  - правила для Си, FlexeLint for C/C++
- Parasoft
  - статический анализ на основе правил
  - автоматическая генерация тестов
- PREfix/PREfast
  - C/C++. Windows Driver Foundation
- Coverity
  - C/C++, C#, Java. Thread analyzer for Java.
- Klocwork Insight
  - C/C++, C#, Java

Место для обсуждения